# Building and Using Dashboards

Version 9.10

August 2016

APAMA

This document applies to Apama Version 9.10 and to all subsequent releases.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.



**Document ID: PAM-DAS-910-20160810**

# Table of Contents

**About this Guide....17**
Documentation roadmap....17
Online Information....19
Contacting customer support....20

**Building Dashboard Clients....21**
Introduction to Building Dashboard Clients....23
Web client requirements....24
About dashboards....24
Starting the Dashboard Builder....25
Starting Builder from the Windows Start menu....25
Starting Builder from Software AG Designer....26
Specifying Dashboard Builder options....26
Starting Builder from the command line....26
Scenario instance and DataView item ownership....27
Using the tutorial application....27
Using Dashboard Builder....29
Dashboard Builder layout....30
The menubar....30
The toolbar....34
The canvas....35
The Object Palette....35
The Object Properties panel....35
Specifying data sources....37
Specifying correlators....37
Specifying XML data sources....38
Activating data source specifications....38
Saving data source specifications....38
Setting the background properties....38
About resize modes....39
About resize modes and Display Server deployments....42
About resize modes and composite objects....42
Working with objects....43
Adding objects to a dashboard....43
Selecting an object....43
Resizing objects....43
Moving objects....44
Copy and pasting objects....44
Deleting objects....45
Setting Builder options....45
Setting Dashboard options....45

Setting options in the General tab group....46
Setting options in the General tab....46
Setting options in the Substitutions tab....46
Setting options in the Data Server tab....47
Setting options in the Custom Colors tab....47
Setting options in the Apama tab group....48
Setting options in the SQL tab group....48
Setting options in the XML tab group....49
Saving options....49
Command line options for the Dashboard Builder....49
Restrictions....54
Attaching Dashboards to Correlator Data....55
Dashboard data tables....56
Scenario instance table....57
Scenario trend table....58
Scenario OHLC table....59
Correlator status table....59
Data Server status table....60
Scenario constraint table....60
DataView item table....61
DataView trend table....61
DataView OHLC table....61
SQL-based instance table....61
Definition Extra Params table....62
Setting data options....63
Scenario instance and DataView item ownership....63
Creating a data attachment....64
Using the Attach to Apama dialog....64
Selecting display variables or fields....66
Displaying attached data....66
Filtering data....66
Attaching to constraint data....67
About timestamps....67
Using dashboard variables in attachments....67
About non-substitution variables....68
About drilldown and $instanceId....68
About other predefined substitution variables....69
Using SQL-based instance tables....70
Working with multiple Data Servers....72
Builder with multiple Data Servers....73
Viewer with multiple Data Servers....73
Display Server deployments with multiple Data Servers....74
Applet and WebStart deployments with multiple Data Servers....75
Using table objects....76
Creating a scenario summary table....77

Filtering rows of a scenario summary table....78
Performing drilldowns on tables....79
Specifying drill-down column substitutions....82
Hiding table columns....84
Using pre-set substitution variables for drill down....84
Formatting table data....84
Colorizing table rows and cells....85
Setting column headers....87
Using rotated tables....87
Using pie and bar charts....88
Creating a summary pie or bar chart....89
Using series and non-series bar charts....89
Performing drilldowns on pie and bar charts....90
Using trend charts....91
Creating a scenario trend chart....92
Charting multiple variables....96
Adding thresholds....102
Configuring trend-data caching....104
Rendering trend charts in HTML5....105
Using stock charts....111
Using OHLC values....112
Creating a scenario stock chart....117
Adding overlays....121
Recreating the Stock Chart Overlay sample....123
Generating OHLC values....128
Localizing dashboard labels....129
Localizing dashboard messages....133
Using Dashboard Functions....135
Using built-in functions....136
Creating custom functions....139
Developing a custom-function library....139
Implementing getFunctionDescriptors....139
Implementing evaluateFunction....140
Installing a custom-function library....140
Sample IFunctionLibrary implementation....141
Defining Dashboard Commands....149
Scenario lifecycle....150
Defining commands....150
Using dashboard variables in commands....151
Defining commands for creating a scenario instance....153
Defining commands for editing a scenario instance....155
Supporting deletion of a scenario instance....157
Supporting deletion of all instances of a scenario....159
Using popup dialogs for commands....160
Command options....162

Associating a command with keystrokes....162
Defining multiple commands....163
Creating custom commands....164
Developing a custom-command library....164
Installing a Custom-Command Library....165
Sample ICommandLibrary implementation....165
Apama set substitution command....167
Reusing Dashboard Components....169
Using Object Grids....170
Configuring Object Grids....171
Recreating the Object Grid sample....173
Using Composite objects....174
Creating files to display in composite objects....175
Configuring Composite objects....176
Using substitutions with Composite objects....178
Composite object interactivity....179
Composite object sample....180
Recreating the Composite object sample....181
Using Composite Grids....181
Configuring Composite Grids....182
Composite Grid sample....184
Recreating the Composite Grid sample....184
Using include files....186
Include File sample....187
Recreating the Include File sample....187
Working with multiple display panels....188
About the format of the panels-configuration file....189
Using new tags to configure the panels in a window....189
Configuring panels with accordion controls....190
Configuring static tree navigation panels....191
Configuring tabbed navigation panels....191
Using tab definition files....192
Examples of configuration files for multiple panels....193
Using tree controls in panel displays....193
Creating tree controls....194
Configuring tree control icons....200
Specifying tree control properties....205
Tree control limitations....213
Using old tags to configure the panels in a window....213
Using border panels....214
Using card panels....216
Using grid panels....216
Using tabs panels....217
Using tree panels....218
Using the RTViewNavTreePanel tag....221

Using the RTViewPanel tag....222
Sending Events to Correlators....225
Using the Define Apama Command dialog....226
Command field....226
Package field....227
Event field....227
Channel field....227
Other dialog fields....227
Default values....229
Specifying values for complex types....230
Updating event definitions in Builder....230
Example....230
Send event authorization....232
Using XML Data....235
XML data format....236
Scalar data elements....236
Tabular data elements....237
Defining an XML data source....238
Attaching objects to XML data....239
Using SQL Data....243
SQL system requirements and setup....244
Attaching visualization objects to SQL data....244
Validation colors....247
Substitutions....248
Select table columns....248
Defining SQL commands....249
Validation colors....250
Special values....251
Specifying application options....251
SQL tab....251
Adding a Database....252
Database repository....253
Entering database information directory into OPTIONS.ini....254
Generating encrypted passwords for SQL data sources....254
Deploying applet and WebStart dashboards....255
Setting up SQL database connections....255
Direct JDBC connection....256
Setting SQL data source options....257

**Dashboard Property Reference for Graphs, Tables and Trends....259**
Introduction to Dashboard Properties....261
Objects for complex-data visualization....262
About the Object Properties window....262
Editing property values....262
Copying and pasting property values....263

Graph Objects....265
Bar graphs....266
Bar graph: Alert group....267
Bar graph: Background group....273
Bar graph: Bar group....276
Bar graph: Column group....279
Bar graph: Data group....279
Bar graph: Data Format group....282
Bar graph: Data Label group....283
Bar graph: Historian group....285
Bar graph: Interaction group....285
Bar graph: Label group....289
Bar graph: Layout group....290
Bar graph: Legend group....292
Bar graph: Marker group....293
Bar graph: Object group....294
Bar graph: Plot Area group....296
Bar graph: Trace group....297
Bar graph: X-Axis group....299
Bar graph: Y-Axis group....299
Google map....301
Map graph: Data group....303
Map graph: Interaction group....304
Map graph: Map group....305
Map graph: Object group....306
Licensing....309
Heat map....309
Heat maps with one index column....310
Heat maps with multiple index columns....310
Mapping from possible aggregation results to colors....311
Drill down displays....311
Object class name....312
Heat map property groups....312
Heat map: Background properties....312
Heat map: Data group....315
Heat map: Data format group....317
Heat map: Data Label group....318
Heat map: Historian group....318
Heat map: Interaction group....318
Heat map: Label group....321
Heat map: Layout group....323
Heat map: Node group....324
Heat map: Object group....325
Heat map: Quality group....327
Legend....329

Legend: Background group....329
Legend: Data group....332
Legend: Historian group....332
Legend: Interaction group....332
Legend: Label group....334
Legend: Legend group....335
Legend: Object group....337
Pie graph....338
Pie graph: Background group....339
Pie graph: Column group....342
Pie graph: Data group....342
Pie graph: Data Format group....343
Pie graph: Data Label group....344
Pie graph: Historian group....344
Pie graph: Interaction group....345
Pie graph: Label group....348
Pie graph: Legend group....349
Pie graph: Object group....349
Pie graph: Wedge group....351
Radar graph....353
Radar graph: Alert group....354
Radar graph: Background group....359
Radar graph: Column group....362
Radar graph: Data group....362
Radar graph: Data Format group....364
Radar graph: Data Label group....365
Radar graph: Historian group....366
Radar graph: Interaction group....366
Radar graph: Label group....369
Radar graph: Legend group....370
Radar graph: Marker group....371
Radar graph: Object group....371
Radar graph: Plot Area group....373
Radar graph: Radial Axis group....374
Radar graph: Trace group....375
Radar graph: Value Axis group....377
XY graph....379
XY graph: Alert group....380
XY graph: Background group....387
XY graph: Column group....390
XY graph: Data group....390
XY graph: Data Format group....392
XY graph: Data Label group....393
XY graph: Historian group....394
XY graph: Interaction group....394

XY graph: Label group....398
XY graph: Legend group....399
XY graph: Marker group....400
XY graph: Object group....401
XY graph: Plot Area group....403
XY graph: Trace group....404
XY graph: X-Axis group....406
XY graph: Y-Axis group....407
Table Objects....411
Standard tables....412
Standard table: Alert group....413
Standard table: Background group....415
Standard table: Cell group....418
Standard table: Column group....419
Standard table: Column Header group....423
Standard table: Data group....424
Standard table: Data Label group....425
Standard table: Grid group....425
Standard table: Historian group....426
Standard table: Interaction group....426
Standard table: Label group....430
Standard table: Object group....431
Standard table: Row Header group....433
Standard table: Sort group....434
Rotated tables....435
Rotated table: Background group....436
Rotated table: Cell group....438
Rotated table: Column group....439
Rotated table: Data group....441
Rotated table: Grid group....441
Rotated table: Historian group....441
Rotated table: Interaction group....442
Rotated table: Label group....444
Rotated table: Object group....445
HTML5 tables....447
Trend Objects....453
Sparkline charts....454
Sparkline chart: Alert group....454
Sparkline chart: Background group....457
Sparkline chart: Data group....460
Sparkline chart: Data Format group....461
Sparkline chart: Historian group....461
Sparkline chart: Interaction group....461
Sparkline chart: Label group....464
Sparkline chart: Legend group....465

Sparkline chart: Marker group....467
Sparkline chart: Object group....467
Sparkline chart: Plot Area group....469
Sparkline chart: Trace group....469
Sparkline chart: X-Axis group....471
Sparkline chart: Y-Axis group....472
Stock charts....472
Stock chart: Background group....473
Stock chart: Data group....476
Stock chart: Data Format group....476
Stock chart: Interaction group....476
Stock chart: Historian group....480
Stock chart: Label group....480
Stock chart: Legend group....481
Stock chart: Object group....483
Stock chart: Plot Area group....484
Stock chart: Price Trace group....485
Stock chart: Trace group....489
Stock chart: TraceN group....490
Stock chart: X-Axis group....492
Stock chart: Y-Axis group....495
Trend graphs....497
Trend graph: Alert group....499
Trend graph: Background group....504
Trend graph: Data group....506
Trend graph: Data Format group....508
Trend graph: Interaction group....508
Trend graph: Label group....512
Trend graph: Legend group....513
Trend graph: Marker group....514
Trend graph: Object group....515
Trend graph: Plot Area group....517
Trend graph: Trace group....518
Trend graph: TraceN group....520
Trend graph: Trace Groups group....527
Trend graph: X-Axis group....528
Trend graph: Y-Axis group....531
Drill-Down Specification....535
Using the Drill Down Properties dialog....536
Activating drill downs....538
About drilldown displays opened in Dashboard Builder....538

**Dashboard Function Reference....541**
Introduction to Dashboard Functions....543
Working with functions....544

Scalar Functions....547
Add....549
Average....549
Boolean Expression....549
Concatenate....550
Correlator Time Format....550
Date Add....551
Date Ceiling....551
Date Compare....552
Date Difference....552
Date Floor....553
Date Format....553
Date Now....554
Delta....554
Divide....554
Duration....555
Evaluate Expression As Double....555
Evaluate Expression As String....556
Format Number....556
Get Substitution....557
Init Local Variable....557
isWindowsOS....557
Max....557
Min....558
Modulo....558
Multiply....558
Percent....558
Quick Set Sub....559
Replace All....559
Replace Value....560
Set Substitution....560
Set Substitutions By Lookup....560
Subtract....561
Validate Substitutions....561
Tabular Functions....563
Add All Rows Or Columns....565
Add Columns....566
Average All Rows Or Columns....567
Average Columns....568
Baseline Over Time....569
Buffer Table Rows....571
Combine....572
Concatenate Columns....573
Convert Columns....574
Copy....575

Count....576
Count By Bands....576
Count Unique Values....577
Count Unique Values By Time....577
Create Selector List....578
Delta And Rate Rows....579
Delta Rows....579
Distinct Values....580
Divide Columns....581
Ensure Columns....582
Ensure Timestamp Column....582
Evaluate Expression By Row....583
Filter And Extract Matches....585
Filter By Pattern....586
Filter By Row....586
Filter By Time Range....587
First Table Rows....587
Format Table Columns....588
Get Data Server Connection Status....588
Group By Time....589
Group By Time and Unique Values....590
Group by Unique Values....591
Join....596
Join Outer....598
Last Table Rows....599
Mark Time Gaps....599
Max All Rows or Columns....600
Max Columns....601
Min All Rows or Columns....602
Min Columns....603
Modulo Columns....604
Multiply Columns....605
Percent Columns....606
Pivot On Unique Values....607
Reference....609
Rename Columns....609
Select Column....610
Set Column Type....610
Sort Table....611
Split String....611
String to Table....611
Subtotal By Time....612
Subtotal By Unique Values....613
Subtract Columns....614
Table Contains Values....615

Expression Syntax in Dashboard Functions....617
Operators in dashboard function expressions....618
Arithmetic functions in dashboard function expressions....618
String functions in dashboard function expressions....621

**Dashboard Deployment....623**
Dashboard Deployment Concepts....625
Deployment options....626
Application installation....626
Authentication....626
Authorization....626
Data protection....627
Refresh latency....627
Scalability....627
Dashboard support for Apple iPad....627
Data server and display server....628
Process architecture....628
Builders and administrators....631
Generating Dashboards....633
Starting the wizard....634
Using the wizard....635
Using the titlebar/toolbar....635
Using the Introduction form....636
Using the Main, Create, Edit, and Details Forms....636
Using the layout configuration forms....637
Preparing Dashboards for Deployment....639
Dashboard feature checklist....640
Changing correlator definitions for deployment....641
Choosing among deployment types....641
Application installation....641
Authentication....642
Authorization....642
Data protection....642
Scalability....642
Choosing among Web-based deployment types....643
Installation....643
Served data....643
Refresh latency....643
Dashboard command support....643
Dashboard iPad Support....643
Using the Deployment Configuration editor....643
Starting the Configuration editor....644
Saving deployment configurations....644
Title bar/Toolbar....644
General Settings....645

Startup and Server....645
Additional JAR Files....645
Layout....646
Using the Dashboard Package wizard....646
Generating a deployment package from the command line....647
Sharing information with the Dashboard Administrator....648
Deploying Dashboards....649
Generating the dashboard .war file....650
Installing a dashboard .war file....650
Inside a dashboard .war file....651
Additional steps for display server deployments....651
Applet and WebStart Deployment....652
Managing the Dashboard Data Server and Display Server....653
Prerequisites....654
Starting and stopping the Data Server or Display Server....654
Command line options for the Data Server and Display Server....655
Controlling Update Frequency....662
Configuring Trend-Data Caching....663
Managing Connect and Disconnect Notification....667
Working with multiple Data Servers....667
Builder with multiple Data Servers....668
Viewer with multiple Data Servers....669
Display Server deployments with multiple Data Servers....670
Applet and WebStart deployments with multiple Data Servers....671
Managing and stopping the Data Server and Display Server....672
Administering Dashboard Security....677
Administering authentication....678
Authentication for local and WebSphere deployments....678
Dashboard Login Modules Provided by Apama....679
Developing custom login modules....679
Installing login modules....679
Installing UserFileLoginModule....681
Installing LdapLoginModule....681
Administering authorization....682
Users and roles....683
Default Scenario and DataView access control....683
Customizing Scenario and DataView access control....684
Providing a Scenario Authority....684
Developing a custom Scenario Authority....684
Implementing IScenarioAuthority methods....684
Installing a Scenario Authority....686
Sample Custom Scenario Authority....686
Send event authorization....687
Using UserCredential accessor methods....687
Compiling your Event Authority....687

Installing your Event Authority..... 687
Providing a login module that supports a Scenario or Event Authority..... 688
Securing communications.....689
Example: Implementing LoginModule..... 689

**Using the Dashboard Viewer.....695**
Concepts Underlying Dashboards.....697
About Dashboards.....698
Starting the Dashboard Viewer..... 699
Scenario instance and DataView item ownership.....700
Using the Dashboard Viewer.....701
Opening and viewing dashboards.....702
Running the Statistical Arbitrage demo.....702
Displaying additional dashboards.....702
Creating scenario instances visualized by the Statistical Arbitrage main dashboard.....703
The Dashboard Viewer menu bar.....703
Resizing the Dashboard Viewer.....705
Working with Dashboard Objects.....705
Trend charts..... 705
Stock charts.....706
Tables..... 706
Pie and Bar charts..... 707
Startup Options for the Dashboard Viewer..... 709
Timezone ID Values..... 715

# About this Guide

An Apama dashboard provides the ability to view and interact with scenarios and DataViews. An Apama project typically uses one or more dashboards, which are created in the Dashboard Builder. The Dashboard Viewer provides the ability to use dashboards created in Dashboard Builder. Dashboards can also be deployed as simple Web pages, applets, or WebStart applications. Deployed dashboards connect to one or more correlators via a Dashboard Data Server or Display Server.

## Documentation roadmap

Apama provides documentation in the following formats:

- HTML (viewable in a web browser)
- PDF (available from the documentation website)
- Eclipse help (accessible from the Software AG Designer)

You can access the HTML documentation on your machine after Apama has been installed:

- **Windows.** Select **Start > All Programs > Software AG > Tools > Apama *n.n* > Apama Documentation *n.n***. Note that **Software AG** is the default group name that can be changed during the installation.
- **UNIX.** Display the `index.html` file, which is in the `doc` directory of your Apama installation directory.

The following table describes the different guides that are available.

| Title | Description |
|---|---|
| *Release Notes* | Describes new features and changes since the previous release. |
| *Installing Apama* | Instructions for installing Apama. |
| *Introduction to Apama* | Introduction to developing Apama applications, discussions of Apama architecture and concepts, and pointers to sources of information outside the documentation set. |
| *Using Apama with Software AG Designer* | Instructions for using Apama to create and test Apama projects, develop EPL programs, define Apama queries, develop JMon programs, and store, retrieve and play back data. |

| Title | Description |
| --- | --- |
| *Developing Apama Applications* | Describes the different technologies for developing applications: EPL monitors, Apama queries, Event Modeler, and Java. You can use one or several of these technologies to implement a single Apama application. In addition, there are C++, C, and Java APIs for developing components that plug in to a correlator. You can use these components from EPL. |
| *Connecting Apama Applications to External Components* | Describes how to connect Apama applications to any event data source, database, messaging infrastructure, or application. The general alternatives for doing this are as follows:<br>■ Implement standard Apama Integration Adapter Framework (IAF) adapters.<br>■ Create applications that use correlator-integrated messaging for JMS or Software AG's Universal Messaging.<br>■ Use connectivity plug-ins written in Java or C++.<br>■ Develop adapters with Apama APIs for Java and C ++.<br>■ Develop client applications with Apama APIs for Java, .NET, and C++. |
| *Building and Using Dashboards* | Describes how to build and use an Apama dashboard, which provides the ability to view and interact with scenarios and DataViews. An Apama project typically uses one or more dashboards, which are created in the Dashboard Builder. The Dashboard Viewer provides the ability to use dashboards created in Dashboard Builder. Dashboards can also be deployed as simple Web pages, applets, or WebStart applications. Deployed dashboards connect to one or more correlators by means of a Dashboard Data Server or Display Server. |
| *Deploying and Managing Apama Applications* | Describes how to deploy components with Command Central or with Apama's Enterprise Management and Monitoring (EMM) console. Also provides information for: |

| Title | Description |
| --- | --- |
| | ■ Improving Apama application performance by using multiple correlators and saving and reusing a snapshot of a correlator's state.<br>■ Managing and monitoring over REST (Representational State Transfer).<br>■ Using correlator utilities. |

In addition to the above guides, Apama also provides the following API reference information:

- API Reference for EPL in ApamaDoc format
- API Reference for Java in Javadoc format
- API Reference for C++ in Doxygen format
- API Reference for .NET in HTML format

# Online Information

### Software AG Documentation Website

You can find documentation on the Software AG Documentation website at http://documentation.softwareag.com. The site requires Empower credentials. If you do not have Empower credentials, you must use the TECHcommunity website.

### Software AG Empower Product Support Website

You can find product information on the Software AG Empower Product Support website at https://empower.softwareag.com.

To submit feature/enhancement requests, get information about product availability, and download products, go to Products.

To get information about fixes and to read early warnings, technical papers, and knowledge base articles, go to the Knowledge Center.

### Software AG TECHcommunity

You can find documentation and other technical information on the Software AG TECHcommunity website at http://techcommunity.softwareag.com. You can:

- Access product documentation, if you have TECHcommunity credentials. If you do not, you will need to register and specify "Documentation" as an area of interest.
- Access articles, code samples, demos, and tutorials.

- Use the online discussion forums, moderated by Software AG professionals, to ask questions, discuss best practices, and learn how other customers are using Software AG technology.
- Link to external websites that discuss open standards and web technology.

## Contacting customer support

If you have an account, you may open Apama Support Incidents online via the eService section of Empower at https://empower.softwareag.com/. If you do not yet have an account, send an email to empower@softwareag.com with your name, company, and company email address and request an account.

If you have any questions, you can find a local or toll-free number for your country in our Global Support Contact Directory at https://empower.softwareag.com/public_directory.asp and give us a call.

# I Building Dashboard Clients


- Introduction to Building Dashboard Clients ..... 23
- Using Dashboard Builder ..... 29
- Attaching Dashboards to Correlator Data ..... 55
- Using Dashboard Functions ..... 135
- Defining Dashboard Commands ..... 149
- Reusing Dashboard Components ..... 169
- Sending Events to Correlators ..... 225
- Using XML Data ..... 235
- Using SQL Data ..... 243

*Building Dashboard Clients* assumes that you have already installed a development version of Apama. You should also read *Introduction to Apama* in the Apama documentation set.

# 1 Introduction to Building Dashboard Clients


- Web client requirements ..... 24
- About dashboards ..... 24
- Starting the Dashboard Builder ..... 25
- Scenario instance and DataView item ownership ..... 27
- Using the tutorial application ..... 27

This chapter introduces dashboards and the Dashboard Builder. It also describes how to run the tutorial application that is a companion to *Building Dashboard Clients*.

## Web client requirements

Web clients can access Apama dashboards through their web browser. No Apama installation is required for web clients but they must satisfy the following requirements:

- A supported web browser version. See the *Supported Platforms* document for the current Apama version. This is available from the following web page: http://documentation.softwareag.com/apama/index.htm.
- Oracle's Java plug-in must be installed for dashboards deployed either as applets or through Java Web Start technology. There is no Java plug-in requirement for dashboards deployed with the Apama Dashboard Display Server.
- Your browser must have cookies enabled for web pages served from the host where you are running your application server.
- If you have a pop-up blocker, be sure to set it to allow pop-ups for web pages served from the host where you are running your application server.

## About dashboards

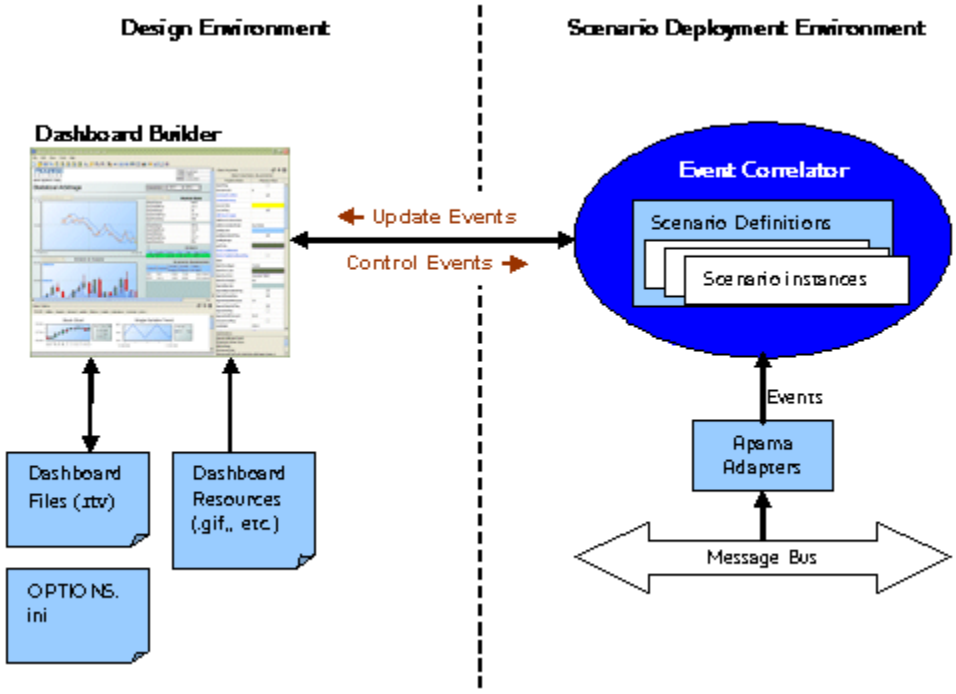
Dashboards provide the ability to view and interact with scenarios and DataViews. They contain charts and other objects that dynamically visualize the values of scenario variables and DataView fields. Dashboards can also contain control objects for creating, editing, and deleting scenario instances and DataView items.

An Apama project typically uses of one or more dashboards. Each dashboard defines a single display, or view, of information. Dashboards are created in the Dashboard Builder or with the Dashboard Generation Wizard in Software AG Designer (see "Generating Dashboards" on page 633). Each dashboard is stored in a separate `.rtv` file. All `.rtv` files for a given project are stored in a single directory. This directory also contains a `.dashboard` file, which records various dashboard parameters, including the file that is to be used as the dashboard project's *main* dashboard.

The contents of a dashboard, the charts displayed and the data shown, is determined when the dashboard is created in the Builder. The Dashboard Viewer provides the ability to use dashboards created in the Builder. Dashboards can also be deployed as simple Web pages, applets, or WebStart applications.

Deployed dashboards connect to one or more correlators via a Dashboard Data Server or Display Server. As the scenarios in a correlator run and their variables change, or as a DataView item's fields are updated, update events are sent to all connected dashboards. When a dashboard receives an update event, it updates its display in real time to show the behavior of the scenarios or DataViews. User interactions with the dashboard, such as creating an instance of a scenario, result in control events being sent via the Data Server or Display Server to the correlator.

Dashboard Builder communicates with running correlators so that you can see at design time what a dashboard will look like when deployed. Unlike a deployed dashboard, the Builder connects directly to the correlators it communicates with. The following diagram illustrates the design environment for dashboards:




In order to use Dashboard Builder to create a dashboard for a scenario or DataView, you need to start a correlator and inject the scenario or DataView into it. You should use a development correlator to initially develop dashboards, not a deployed correlator acting on live data.

Dashboard Builder does not support creating dashboards for scenarios or DataViews that have not been injected into a correlator.

## Starting the Dashboard Builder

You can start the Dashboard Builder from the Windows Start menu, from Software AG Designer, or from the command line.

### Starting Builder from the Windows Start menu

The simplest way to start the Dashboard Builder is from the Windows **Start** menu.

1. Select **All Programs > Software AG > Tools > Apama *n.n* > Apama Dashboard Builder *n.n***.

When you start the Builder this way, the Builder's current directory is the `dashboards` directory in your Apama installation's work directory.

## Starting Builder from Software AG Designer

You can use Software AG Designer to open a specified file in the Builder.

1. Do one of the following:
   - Double-click a dashboard file in the navigation pane.
   - Right-click a dashboard file in the navigation pane, and select **Open With > Apama Dashboard Builder**.
   - Select **File > Open File...**. In the Open File dialog, navigate to a dashboard file or enter a pathname, and then click **OK**.

When you start the Builder this way, the Builder's current directory is the directory that contains the opened file.

#### *Specifying Dashboard Builder options*

You can specify the options that will be used when an Apama project opens Dashboard Builder. The options correspond to the command line arguments available when you manually start the Dashboard Builder executable. These options are described in "Command line options for the Dashboard Builder" on page 49.

1. In the **Project Explorer**, right-click the project and select **Properties** from the pop-up menu. (You can also select **Project > Properties** from the menu bar.)

   The Properties dialog is displayed.
2. In the left panel, expand **Apama** and select **Dashboard Properties**.
3. In the Dashboard Properties panel on the right, select the **Dashboard Builder Options** tab.
4. On the **Dashboard Builder Options** tab, in the **Dashboard command line arguments** field, specify the desired options. Multiple options should be entered on a single line.
5. Click **OK**.

## Starting Builder from the command line

Dashboard Builder can be started by running the `dashboard_builder` executable located in the `bin` directory of your Apama installation. This method of starting the Builder allows you to pass startup options on the command line. The Builder startup options are detailed in "Command line options for the Dashboard Builder" on page 49.

**To run the Builder from the command line**

1. Do one of the following:
   - Use an Apama Command Prompt as described in *Deploying and Managing Apama Applications* in the topic *Setting up the environment using the Apama Command Prompt*.

- Set your current directory to the Apama `bin` directory.

## Scenario instance and DataView item ownership

Scenario instances and DataView items in a correlator include an attribute identifying the owner of the instance. When a scenario instance is created through a dashboard, it provides the current user ID as the owner of the instance.

By default, you are only allowed to see and operate on those scenario instances and DataView items that you own, that is, the current user ID must match the owner attribute of the instance. There is one exception to this default: if the owner is specified as "`*`", all users have access by default.

## Using the tutorial application

This guide contains numerous examples that are bundled as a tutorial with your Apama installation. Use this tutorial in the Dashboard Builder while you read this guide. Many sections in this guide instruct you to create or modify dashboards. This “learning by doing” approach is the philosophy behind this guide.

The Dashboard Builder connects to one or more correlators in order to discover the scenarios and DataViews that the correlators have loaded. Information about these scenarios and DataViews is then made available for use in the design of the dashboard.

Follow these steps to run the tutorial:

1. In Software AG Designer, go to Apama's **Workbench** perspective, and select **File > Import**.
2. In the **Import** dialog, expand **General**, select **Existing Projects into Workspace** and click **Next**.
3. Next to the **Select root directory** field, click the **Browse** button, navigate to the `samples` directory in your Apama installation directory, select the `dashboard_studio` folder. Click **OK**.
4. Make sure the **Dashboard Tutorial** project is selected.
5. Select **Copy projects into workspace** and click **Finish**.
6. In the **Workbench Project View**, select and display the **Dashboard Tutorial** project.
7. Click the **Start**  button.

Software AG Designer injects the necessary EPL and tutorial scenario into the correlator. In addition, it creates instances of the scenario. The instances of the scenario running in the correlator provide event data that is displayed in the dashboard. After a few moments, the Dashboard Builder appears. The tutorial is configured to automatically open the tutorial main page, which is defined in the file `tutorial-main.rtv` in `samples\dashboard_studio\tutorial\dashboards` under your Apama installation directory.

Double-clicking a topic displayed on the tutorial main page displays a page providing an example of the topic. The tutorial uses the tutorial scenario located in the `monitors` folder in the `tutorial` directory. This is a very simple scenario designed for with this guide.

Instances of the scenario are created by specifying values for the input variables **Instrument** and **Clip Size**. The scenario uses a simulated market feed to drive changes in the price of the instrument. The scenario tracks the **Velocity** of the **Price** and issues a simulated trade every second based on the **Velocity** being positive or negative. On each trade the number of shares specified as the **Clip Size** are bought or sold.

The scenario has six variables.

- **Instrument**: The name of the instrument being traded
- **Clip Size**: The number of shared to trade on each order
- **Price**: The current price of the instrument
- **Velocity**: The current velocity on the instrument's price
- **Shares**: The number of shares currently held of the instrument
- **Position**: The total position in the instrument.

Dashboard Builder provides a large set of objects with a wide range of configurable properties. This variety enables you to create visually rich and flexible dashboards which best meet the needs of your applications and users. This guide does not detail every object and every property. Rather, it presents the most commonly used objects and how they are used.

The information presented in this guide enables you to create dashboards for your scenarios and DataViews. You should experiment with the objects and properties not presented in this guide to gain even greater flexibility in your dashboard design.

*Do not save your changes* to the tutorial as changes might make it impossible to use it as a tutorial in subsequent sections of this guide. If you have saved it by mistake, you can restore it from your distribution by re-running the installer.

# 2 Using Dashboard Builder


- Dashboard Builder layout ..... 30
- Specifying data sources ..... 37
- Setting the background properties ..... 38
- About resize modes ..... 39
- Working with objects ..... 43
- Setting Builder options ..... 45
- Setting Dashboard options ..... 45
- Command line options for the Dashboard Builder ..... 49
- Restrictions ..... 54

This chapter illustrates how to use the Dashboard Builder's interactive functionality. Chapter 1 introduced the concepts underlying Apama dashboards. Subsequent chapters detail how to build dashboards.

## Dashboard Builder layout

This section describes each of the panels available in the Dashboard Builder and how to use them effectively.

### The menubar

| Menu | Command | Description |
|---|---|---|
| **File** | | Operations for opening, saving, and closing dashboards. |
| | **New** | Create a new dashboard. |
| | **Open** | Open an existing dashboard file by browsing. |
| | **Save** | Save the dashboard file. |
| | **Save As** | Save the dashboard to a specific file, possibly different to where it has been saved before. |
| | **Background Properties** | Display the **Background Properties** dialog for setting the size and background color or image for the dashboard. |
| | **Print** | Print the current contents of the dashboard. |
| | **Exit** | Exit Dashboard Builder. |
| **Edit** | | Operations for editing and manipulating dashboard objects |
| | **Add** | Displays the Object Palette if not currently displayed. |
| | **Object Properties** | Displays the Object Properties panel if not currently displayed. |

| Menu | Command | Description |
|---|---|---|
| | **Undo** | Un-does the last Builder operation (that has not been undone already). |
| | **Redo** | Re-does the last undone operation (that has not yet been re-done). |
| | **Copy** | Copy the currently selected object into the copy buffer. |
| | **Paste** | Paste the object in the copy buffer onto the dashboard. |
| | **Paste Data Attachments** | Paste the data attachments of the object in the copy buffer onto the selected object. Only properties common to both objects will be pasted onto the selected object. |
| | **Paste Static Properties** | Paste static properties only; do not include those properties that are attached to data. |
| | **Paste All Properties** | Paste all properties, that is, those that are attached to data as well as static properties. |
| | **Align** | Align the specified feature of the currently selected objects. For example, **Align \| Top** aligns the tops of all currently selected objects with one another. The object that was selected first among all the currently selected objects does not move; all other objects are aligned with the first-selected object. **Top**, **Bottom**, and **Center Horizontal** arrange the objects horizontally, one next to the other. **Left**, **Right**, and **Center Vertical** arrange the objects vertically, one above the other. |
| | **Distribute** | Distribute the currently selected objects so that they are spaced evenly, either horizontally or vertically, between the first-selected object and the last-selected one. The first and last objects do not move. |
| | **Order** | Move the selected object in back of or in front of all other dashboard objects. |

| Menu | Command | Description |
|---|---|---|
| | **Select All** | Select all objects on the current dashboard. |
| | **Delete** | Delete the selected object. |
| **View** | | Operations for zooming in and out on the dashboard. |
| | **Zoom In** | Zoom in on a location in the dashboard. This will switch the pointer to zoom mode as indicated by the pointer changing to a crosshair . In this mode you can click an area of the dashboard to zoom in on it; displaying it in greater detail. Right-click to exit zoom mode. |
| | **Zoom Out** | Zoom out on a location in the dashboard. This will switch the pointer to zoom mode as indicated by the pointer changing to a crosshair . In this mode you can click an area of the dashboard to zoom out on it; displaying it in lesser detail. Right-click to exit zoom mode. |
| | **Zoom Rect** | Zoom in on an area in the dashboard. This will switch the pointer to zoom mode as indicated by the pointer changing to a crosshair . In this mode you can click and drag to select an area of the dashboard to zoom in on. Right click to exit zoom mode. |
| | **Pan** | Pan the dashboard to show areas not currently displayed. This will switch the pointer to zoom mode as indicated by the pointer changing to the pan pointer  . In this mode you can click and drag the dashboard to reveal areas not displayed. Right-click to exit pan mode. It is not possible to pan if 100% of the dashboard is visible. |
| | **100%** | Make the entire dashboard visible. |
| **Tools** | | Operations for defining dashboard options and changing preferences. |

| Menu | Command | Description |
|---|---|---|
| | **Options...** | Displays the **Application Options** dialog for defining data sources and setting other runtime options for deployed dashboards. |
| | **Builder Options...** | Displays the **Builder Options** dialog for setting Dashboard Builder, such as grid characteristics. When snap-to-grid is enabled, object can be positioned only along grid lines, which facilitates object alignment and distribution. |
| | **Functions** | Displays the **Functions** panel for defining dashboard functions. |
| | **Variables** | Displays the **Variables** panel for defining dashboard variables. |
| | **Include Files** | Displays the **Include Files** dialog for including dashboard files in the current dashboard. |
| | **Object List** | Displays the **Object List** panel, which lists the name, class name, and position of each object on the current dashboard. |
| | **Preview Window** | Preview the current dashboard, so you can test interactive functionality such text entry. Save your changes to enable this item. |
| | **Pause Display** | Pause the automatic update of the dashboard. When not paused, the dashboard automatically updates as data changes; when paused, the dashboard does not. When paused, clicking on the dashboard causes it to update. |
| | **Reset Window Layout** | Reset window size and panels to their default configuration. |
| **Help** | | Information about Apama and Dashboard Builder. |
| | **Help Contents** | Opens the Apama documentation to the Introduction in *Building Dashboard Clients*. |

| Menu | Command | Description |
|---|---|---|
| | **Command Line Options** | Displays a list of the Builder options that you can supply at the command line. |
| | **About** | Displays information about this version of the Dashboard Builder. |

## The toolbar

The toolbar contains a number of icons that correspond to commonly used operations. Note that all the operations accessible from the toolbar are also available on the menu bar. The operations are:

| Button | Description |
|---|---|
|  | Create a new dashboard file. |
|  | Open an existing dashboard file. |
|  | Save the current dashboard file. |
|  | Preview the current dashboard. Save your changes to enable this tool. |
|  | Copy the selected object to the copy buffer. |
|  | Paste the object in the copy buffer onto the dashboard. |
|  | Paste the data attachment properties. |
|  | Paste the static properties. |
|  | Paste all properties. |
|  | Delete. |
|  | Undo. |
|  | Redo. |

| Button | Description |
|---|---|
|  | Show or hide grid. |
|  | Enable or disable snap to grid. |
|  | Select by extent. |
|  | Zoom in on the dashboard. |
|  | Zoom out on the dashboard. |
|  | Display the Object Palette. |
|  | Display the Object Properties panel. |
|  | Display the Application Options dialog. |

## The canvas

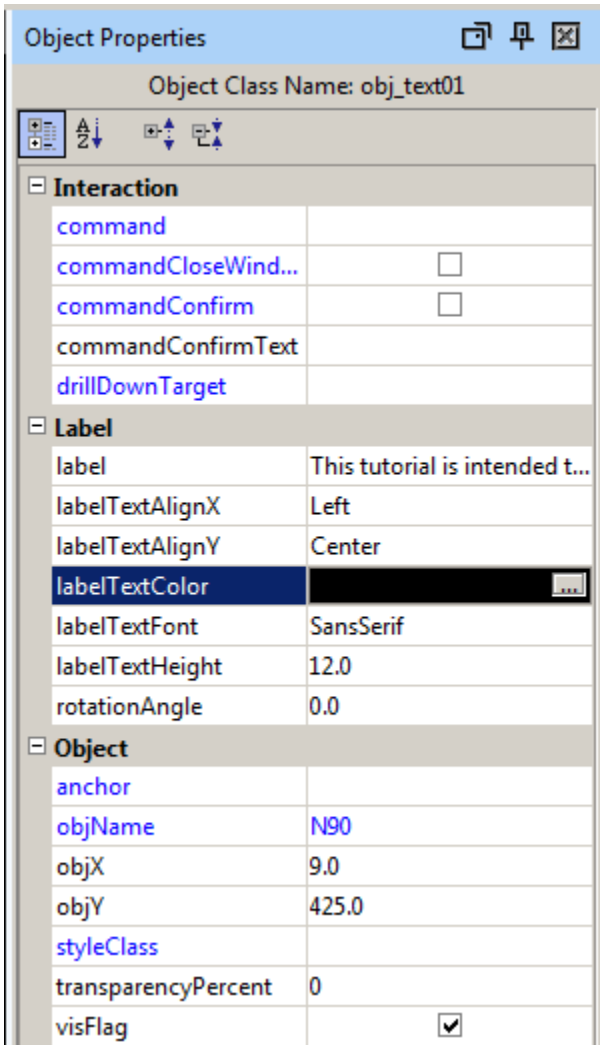
The canvas is where you lay out the objects for a dashboard. Objects can be added to the canvas, moved, and resized. As objects are attached to data sources, the objects will update in real time to reflect changes in the data. This allows you to see how the objects will appear when the dashboard is deployed.

## The Object Palette

The Object Palette presents all object types that may be added to a dashboard. It is organized into separate tabs for different categories of objects.

## The Object Properties panel

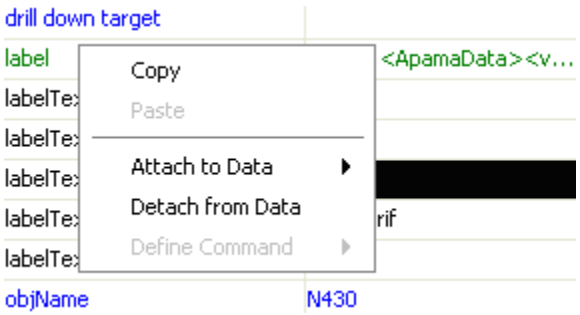
The Object Properties panel displays the properties and their values for the selected object on the canvas. If no object is selected, the properties panel is empty. The set of properties displayed depends on the type of object selected.




The type of object is identified following the **Object Class Name** label at the top of the properties panel; in this case the type is **obj_text01**.

To edit a property, click the property value. Some properties allow you to type in a value, some provide a drop down list of choices, and some present a "..." button for displaying a dialog for setting the property value.

Right-click a property name to display a menu for the property, for example:



Property values can be copied and pasted onto other properties. Properties can also be attached to data sources as detailed in subsequent chapters.

Properties are color coded as follows:

- Blue indicates a static property that cannot be attached to data.
- Green indicates a property that has been attached to data.
- Black indicates a property that may be attached to data.

## Specifying data sources

Dashboard Builder supports building dashboards that display data for scenarios or DataViews in a correlator as well as data from a properly formatted XML file.

To use a correlator or an XML file, you need to make it a known data source to the Dashboard Builder. The topics below detail how to define data sources in the Builder.

See also "Using SQL Data" on page 243 for information on JDBC data sources.

### Specifying correlators

You must specify the correlator in which the scenario or DataView is running before you create a dashboard.

**To create a dashboard for a scenario or DataView in the Dashboard Builder**

1. From the **Tools** menu select **Options**.
2. In the tab list on the left of Applications Options dialog select **Apama**.

   The **Correlator** subtab displays the correlators known to the Builder. Initially only the default correlator for the localhost will be known. For each correlator the following fields are specified

   - **Logical name** – The name that will be used to refer to the correlator. This name cannot be changed once a correlator has been added.
   - **Host** – The host of the correlator. (*Note:* Non-ascii characters are not allowed in host names.)
   - **Port** – The port of the correlator.
   - **Raw channel** – Option to use the raw channel for communication with the correlator. By default the data channel is used.
3. Select the entry for localhost and click the **Edit** button.

   The Correlator Properties dialog allows you to specify the properties of a correlator.
4. Click **Cancel** to close the Correlator Properties dialog.

If you are using the tutorial dashboard, do not change the properties of the default correlator unless you have loaded the tutorial scenario in a correlator running on a different host or on a different port.

5. Use the **Add** button to add a new correlator and the **Delete** button to delete the selected correlator.

### Specifying XML data sources

Dashboard Builder enables you to augment your dashboard by using XML data files as a data source in addition to Apama scenarios and DataViews. The properties of dashboard objects can be attached to data elements in XML files. See "Using XML Data" on page 235, for details on using XML data sources.

### Activating data source specifications

**To activate data source specifications**

1. In the Application Options dialog, click the **OK** or **Apply** to make any changes active for the current invocation of the Builder. This does not save them for future invocations.

### Saving data source specifications

**To save data source specifications**

1. Click the **Save** button to save options settings including data source definitions as detailed in section "Saving options" on page 49.

## Setting the background properties

Background properties control the size, color, and an optional background image for a dashboard.

**To set background properties**

1. Select **Background Properties** from the **File** menu in the menu bar.

   The Background Properties dialog appears.

2. Set the fields **Model Width** and **Model Height** to specify the size of the dashboard. If the dashboard is made smaller than its current size, and the resize mode is **crop** (see below), one or more objects may no longer be visible.

3. Click on **Model Properties**.

   The **Object Properties** panel displays additional properties for the dashboard background.

Use `resizeHeightMin` and `resizeWidthMin` to set the minimum display size. For Web-based dashboards, scrollbars are present when the size is below the minimum. In Viewer, dashboards cannot be resized below the minimum.

4. To use an image as the background for a dashboard, check the **Use Background Image** check box, and either type in the relative path name to a `.gif`, `.jpg`, or `.png` image file or select an image file from the **Image Name** drop down list.

   The drop down list includes Image files that are located either in the same directory as the dashboard or in a subdirectory of the dashboard's directory.

   **Note:** If an image is used as the background for a dashboard, the dashboard is resized to the dimensions of the image, and the **Model Width** and **Model Height** fields are disabled.

5. To set a non-default resize mode, select an item from the **Resize Mode** drop down list. Resize modes are explained in "About resize modes" on page 39.
6. To set a non-default number of grid rows (which is used in **Layout** resize mode when an object's `dock` property is set to **Fill**), enter a value greater than 1 in the **Dock Fill Rows** field. See "About resize modes" on page 39 for detailed information.
7. To set a non-default number of grid columns (which is used in **Layout** mode when an object's `dock` property is set to **Fill**), enter a value greater than 0 in the **Dock Fill Columns** field. See "About resize modes" on page 39 for detailed information.

## About resize modes

The Dashboard facility supports three window resize modes:

- **Layout**: When a window in this mode is resized, the display is resized to fit the available space. The objects in the display are laid out according to their `anchor` and `dock` properties (see below). The window is not forced to maintain its aspect ratio. Objects that are not docked or anchored move relative to their offset from the top left corner of the display. For example, if the object is centered on the display, the object moves 50% of the resize amount. If the object is centered at 3/4 of the display, it moves 75% of the resize amount. Use this mode for dashboards targeted for the Apple iPad.
- **Scale**: This is the default for the Dashboard Builder and Dashboard Viewer, as well as for all Web deployed dashboards. When a window in this mode is resized, the display and all of the objects in it are scaled to fit the available space. The window is forced to maintain its aspect ratio.
- **Crop**: This is the default for Display Server deployments. When a window in this mode is resized, the display stays the same size. If the window is bigger than the display, empty space appears around the display. If the window is smaller than the display, scrollbars appear. The window is not forced to maintain its aspect ratio.

All three resize modes support zooming the display (right-click and select **Zoom In**, **Zoom Out**, or **Zoom Rect**). In both **Layout** and **Scale** modes, if the window is resized while the display is zoomed, the resize further zooms the display.

In the Dashboard Builder, the window resize modes are only applied to drill down windows. The main window of the Dashboard Builder is always in crop mode.

You can set the window resize mode for each dashboard in the **Background Properties** dialog. If set to **Default**, the application level resize mode (see below) is used. Otherwise, the specified resize mode is used for that display. It is recommended that you set the resize mode on a per-dashboard basis, by using the **Background Properties** dialog.

The application level window resize mode can be set in the **General** tab of the **Application Options** dialog or on the command line with the option -apama.`resizeMode`*mode*, where *mode* is `layout`, `scale`, or `crop`.

If **Default** is selected, the default window resize mode is used. The default is **Crop** for Display Server (thin client) deployments, and **Scale** for applet, WebStart, and local (Dashboard Viewer) deployments, as well as for Dashboard Builder.

If the window resize mode is changed in the **Application Options** dialog in the Dashboard Builder, the new value is only applied to new windows that are opened. Windows that are already open do not change modes.

Two new properties have been added to the **Object** group of all objects in order to support **Layout** mode:

- `dock`: Select **None** (default), **Top**, **Left**, **Bottom**, **Right**, or **Fill**.

  When the dock property on an object is set to one of the sides (**Top**, **Bottom**, **Left**, or **Right**), it is moved to the specified side of the display and stretched to fill that side of the display. If the size of the display changes, the docked objects stretch to fill the available space. For example, if the `dock` property is set to **Top**, the object is moved to the top of the display and the width of the object changes to fill the width of the display. If the display is then made wider, either by changing the **Background Properties** on the display or by resizing the window in **Layout** mode, the width of the object changes to match the new width of the display.

  Multiple objects can be docked to the same side of the display. In this case, the first object is docked against the side of the display, the next object is docked against the edge of the first object, and so on.

  When a display has multiple side-docked objects, the object order controls how the dock layout is applied. The layout is applied to the object list from back to front. For example, if the first object in a display is docked to the top, and the second object is docked to the left, the first object fills the entire width of the display, and the second object fills the left side of the display from the bottom of the first object to the bottom of the display.

  When the `dock` property on an object is set to **Fill**, it fills the available space left in the display after all of the side-docked objects have been positioned. When multiple objects in a display have the `dock` property set to **Fill**, those objects are laid out in a grid in the available space. By default, the grid has one row and as many columns

as objects. You can change the grid rows and columns in the **Background Properties** dialog. If both are set to 0, the default is used. If both the rows and columns are specified, the row value is used and the number of columns is calculated based on the number of objects. If the row value is 0 and the column value is specified, the number of rows is calculated based on the number of objects. The objects are laid out left to right, top to bottom according to the order of the objects in the display. The objects with the dock property set to **Fill** are always laid out after all of the other docked objects.

Once an object is docked, there are some limitations on how you can modify that object in the Dashboard Builder. You cannot move a docked object by dragging or changing `objX` and `objY` in the property sheet. Side-docked objects can only be resized toward the center of the display (for example, if the object is docked to the top of the display, it can only be resized to be taller). Fill-docked objects cannot be resized at all. You cannot resize any docked objects using the `objWidth` or `objHeight` properties in the property sheet. You must drag on the valid resize handle to resize it. It is not moved by **Align** or **Distribute**. Objects can be aligned against a docked object, but the docked object is not moved to align against another object. Docked objects are ignored by **Distribute**.

Note that when an object is docked, the properties `objX`, `objY`, `objWidth`, and `objHeight` may change. For example, suppose that you instantiate a **General** object from the palette, and the properties of the object are as follows: `objX`:250, `objY`:250, `objWidth`:64, and `objHeight`:48. When you set the `dock` property to **Top**, the properties are modified as follows: `objX`:368, `objY`:520, `objWidth`:736, and `objHeight`:48 (no change). If you then change the `dock` property to **Left**, the `objWidth` isn't changed, but the `objHeight` changes so that the object fills the entire height and width of the display. When you change the `dock` setting to **None**, these properties stay the same.

Only objects that support the `objWidth` and `objHeight` properties have the `dock` property.

- `anchor`: Select zero or more of **Top**, **Left**, **Bottom**, and **Right**.

  The `anchor` property is only applied when the display is resized either by changing the **Background Properties** on the display or by resizing the window in **Layout** mode. The anchor property anchors the specified side of the object to the same side of the display. When the display resizes, the number of pixels between the specified side of the object and that side of the display remains constant. If an object is anchored on opposite sides (that is, both **Top** and **Bottom** or both **Left** and **Right**), the object is stretched to fill the available space.

  Only objects that support the `objWidth` and `objHeight` properties support anchoring on opposite sides. If an object has the `dock` property set, the `anchor` property is ignored.

The composite object supports both `dock` and opposing `anchor` sides, but does not behave like other objects if the property `resizeMode` is set to **Size to Display**. In this case, the composite size is controlled by the size of the display that it contains, so any changes

to the width or height of the object result in the composite moving, not resizing. The composite object should not be docked if `resizeMode` is set to **Layout**.

## About resize modes and Display Server deployments

The behavior of thin client, Display Server deployments differs from the description above in the following cases:

When the initial display is opened in the thin client, the browser frame is not resized to match the display size as it is, for example, in the Dashboard Viewer.

In crop mode, the display appears in its full size, and if the browser frame is larger than the display, unused space appears below and to the right of the display. In addition, if the browser frame is smaller than the display, scrollbars appear.

In layout and scale modes, the display briefly appears in its default size, and then resizes to fit the browser frame size. This may also occur if another tab is opened in the browser, the browser is resized, and then the browser tab that contains the thin client is re-opened.

In layout and scale modes, after the browser frame is resized, table objects revert to their original states. For example, if the user clicks on a column header in a table in order to sort the column, after a resize the table reverts to its default sort. Similarly, if the user scrolls in a table or resizes the legend, after a resize the scrollbars and legends revert to their initial position and size.

In scale mode, there is unused space in the browser frame. This is because the display uses the largest four-by-three rectangular area of the frame, to ensure equal scaling in both dimensions. The unused area has the same color as the display background, but does not have a gradient fill.

## About resize modes and composite objects

A new property, `resizeMode`, has been added to the **Composite** category of the composite object. When set to **Size to Display** (the default), the size of the composite is determined by the size of the display that it contains, and the composite cannot be resized. If set to **Layout**, the composite can be resized and the objects in the composite display are laid out according to their `anchor` and `dock` properties.

If `resizeMode` is set to **Layout**, the `dock` and `anchor` properties may be set on the composite so that it resizes during a window resize if the window resize mode is also set to **Layout**. If the window resize mode for the display containing the composite is set to **Scale**, the composite object does a scale instead of a layout.

Note that the `dock` and `anchor` properties should not be setup to stretch the composite object if the `resizeMode` is **Size to Display**. This causes the object to toggle back and forth between stretched and not stretched when the window is resized in **Layout** mode.

# Working with objects

This section details how to lay out a dashboard by adding objects to the canvas and setting their position and size.

## Adding objects to a dashboard

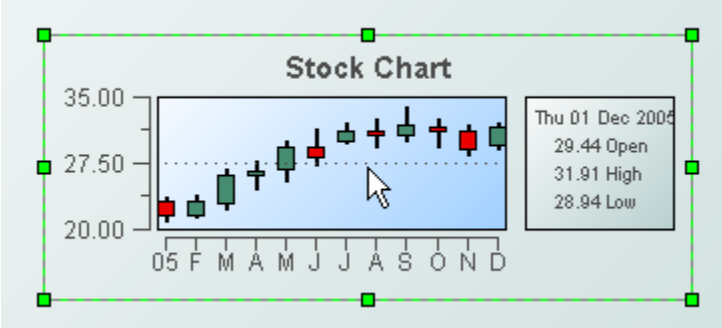
**To add an object to a dashboard**

1. Select the object type that you want to add by clicking on it in the Object Palette.
2. Click on the canvas to add the object.

You can add more objects of the same type by clicking again on the canvas—you don't need to select the same object type again. When you select an object from the Object Palette and then position the cursor over the canvas, the cursor changes to a crosshairs pointer. The appearance of the crosshairs pointer indicates that Builder is in add mode, and clicking will add an object to the canvas. You can adjust the position and size of the object after you have added it.

## Selecting an object

Click an object on the canvas to select it. The selected object is indicated by a rectangle with handles.



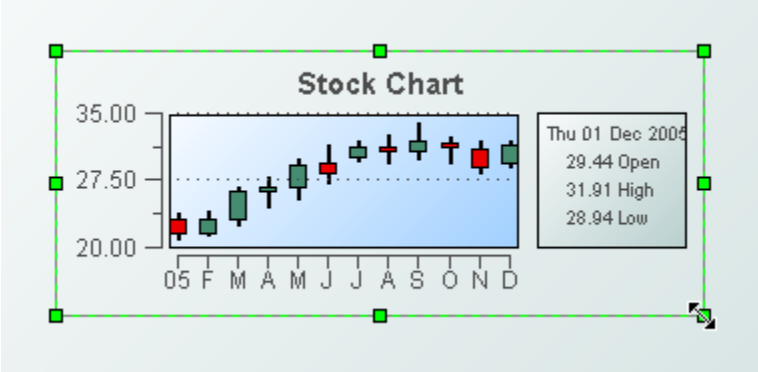

The properties of the selected object are displayed in the Object Properties panel. Actions such as delete operate on the selected object.

To select multiple objects hold down the **Shift** key while clicking on the objects.

**Note:** The Object Properties panel will display the properties of the last selected object.

## Resizing objects

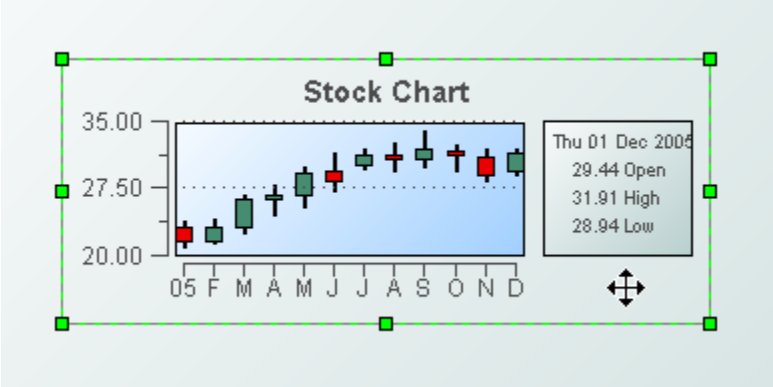
To resize a selected object, drag a handle of the selection rectangle.




You can also set the size of an object by editing the `objWidth` and `objHeight` properties.

## Moving objects

To move a selected object, drag the interior of the selection rectangle.



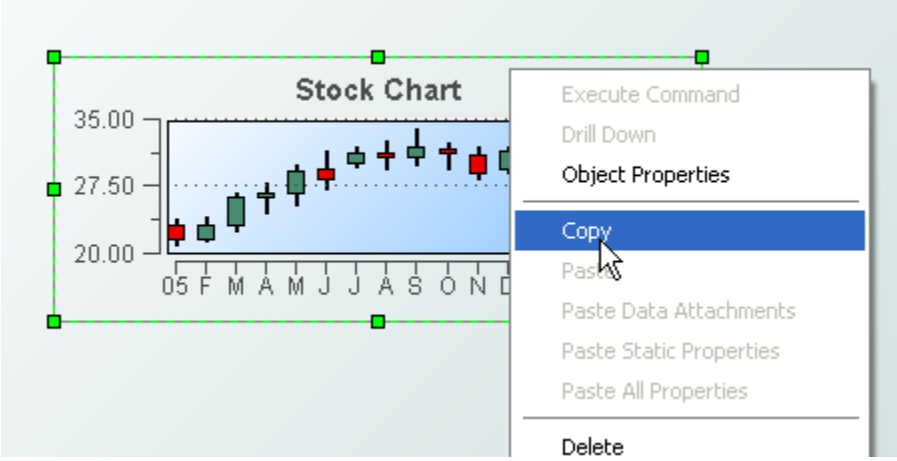

You can also set the position of an object by editing the `objX` and `objY` properties.

The dashboard canvas uses a Cartesian coordinate system, with the origin (0, 0) in the bottom left corner of the dashboard. The `objX` and `objY` properties are relative to the origin.

The `objX` and `objY` properties identify the position of the center of an object. An object positioned at (0, 0) will extend off the left and bottom of the canvas.

## Copy and pasting objects

To copy an object, right-click it to display the object popup menu.




When you select **Copy**, Dashboard Builder places the object into the copy buffer. If the object is already selected, you can also press **Ctrl-C** or select **Copy** from the **Edit** menu in the menu bar.

Once Dashboard Builder has placed an object into the copy buffer, you can add a copy of the object to the canvas by selecting **Paste** from the popup menu (or the **Edit** menu in the menu bar) or by pressing **Ctrl-V**, and then clicking on the canvas. Note that when you select **Paste** or press **Ctrl-V**, the cursor to changes to the **+** pointer.

To copy multiple objects, select each while holding down the **Shift** key and then select **Copy** from a menu or press **Ctrl-C**. When you perform a paste, Dashboard Builder adds a copy of each object to the canvas.

### Deleting objects

To delete an object, right-click the object and then select **Delete** from the popup menu. You can also click it to select it, and then press the **Delete** key or select the **Delete** option from the **Edit** menu in the menu bar. If multiple objects are selected, each will be deleted.

## Setting Builder options

To specify Builder options, select **Builder Options** from the **Tools** menu.

The **Grid** tab allows you to specify properties of the grid that aids layout of visualization objects on the Builder canvas.

The values set in this dialog are automatically restored on application startup and saved on application exit.

## Setting Dashboard options

You can specify dashboard options (user preferences as well as data source definitions) with the Applications Options dialog, described in this section, or with options to the Dashboard Viewer executable (see the *Dashboard Viewer* guide).

To display the Application Options dialog, select **Options** from the **Tools** menu. The Applications Options dialog box appears.

## Setting options in the General tab group

**To set options in the General tab group**

1. Select **General** in the tab group pane (on the left of the dialog).

   The General tab group appears.

***Setting options in the General tab***

1. In the **Update Period** field, enter the rate, in milliseconds, at which the dashboard will refresh. Setting this option to a larger number will reduce the CPU use by the dashboard but at the expense of reducing the frequency with which the dashboard updates.
2. In the **Enable Data** field, check to enable data updates. When data is not enabled, incoming data is ignored.
3. Check the **Redraw After Data Update** check box to specify data-driven redraws.

   Data from an asynchronous data source can arrive at any time between update periods. This means there could be a delay between the time an asynchronous data source receives a data update and when the display showing this data is updated. If selected, displays containing data from asynchronous data sources that have changed since the last update will be redrawn at the rate specified in the **Max Data Redraw Rate** field. Displays where no data has changed will only be redrawn on the update. If not selected, displays are only redrawn based on the update period.
4. In the **Max Data Redraw Rate** field, enter the maximum data redraw rate when data is updated. The default is 500 milliseconds.
5. In the **Confirm Commands** field, set the confirm policy for all command strings. Overrides confirm policies set on individual objects.
6. Check the **Drill Down Windows Always on Top** check box if you want windows displayed as the result of drilldowns to always be on top of their parent window.
7. Check the **Enable Antialiasing** check box to smooth graphics displayed in the dashboard.
8. Check the **Single-Click for Drill Down Commands** to perform drill downs with a single click; not a double-click.
9. In the **Maximum Displays in Composite Object Cache** field, enter the display caching for composite objects.

***Setting options in the Substitutions tab***

The **Substitutions** tab specifies settings that allows substitutions to be added, changed, or deleted.

### *Setting options in the Data Server tab*

If you are an advanced user, the **Data Server** tab allows you to associate a logical name with the Data Server at a given host and port. Advanced users can then use the logical names to specify the Data Server to use for a given attachment or command. (The **Attach to Apama** and **Define Apama Command** dialogs include a Data Server field that can be set to a server's logical name.)

The logical names defined in this tab are used by default for live dashboards viewed with Builder as well as for deployed dashboards. They can be overridden with the `--namedDataServer` option to the builder, viewer, or server executables. See "Working with multiple Data Servers" on page 72 for more information.

Follow these steps to define Data Server logical names:

1. Select **Options** from the **Tools** menu.

   The Applications Options dialog box displays.

2. Select the **Data Server** tab in the **General** tab group.
3. Click the **Add** button to add a definition to the list.

   The **Named Server Configuration** dialog appears.

4. Fill in the dialog fields:

   - **Name:** Logical name of your choosing
   - **Host:** Host of the Data Server whose logical name you are defining
   - **Port:** Port of the Data Server whose logical name you are defining

   To edit or delete a logical-name definition, select the definition in the **Application Options** dialog and click the **Edit** or **Delete** button.

### *Setting options in the Custom Colors tab*

The **Custom Colors** tab allows you to specify custom colors that you can use to set object property values. (You set color-valued object properties with the **Color Chooser** window, which has a **Standard Colors** tab and a **Custom Colors** tab.) Both standard and custom colors are pre-populated when Apama is installed, but you can supplement or modify the custom colors with the **Custom Colors** tab of the **Application Options** dialog.

1. Select **Options** from the **Tools** menu.

   The Applications Options dialog box is displayed.

2. Select the **Custom Colors** tab.
3. Click the **Add** button to add a custom color to the list.

   The **Select Color** dialog appears.

4. To specify a color, select one of the following tabs:

- **Swatches:** Standard Java color palette. Mouse over any swatch to view the RGB values for that color
- **HSV:** Select color choice by hue, saturation, value, and transparency
- **HSL:** Select color choice by hue, saturation, lightness, and transparency
- **HSB:** Color selection by hue, saturation and brightness
- **RGB:** Color selection by red, green and blue intensity
- **CMYK:** Select color by cyan, magenta, yellow, and black intensity well as alpha level

To delete a color, click the **Delete** button.

**Note:** If an object property is defined by a custom color and you delete that color, the color setting for that object property will revert to white.

Apama stores custom colors according to **Color Index** numbers, not RGB values. Therefore if an object property is defined by a custom color and you change the **Color Index** number, the color setting for that object property will revert to white. **Color Index** numbers must be greater than 5000.

To edit a color definition, in the **Color** fields click the **...** button of a selected color to edit that color definition with the **Select Color** dialog.

*Object limitations:* Some objects (for example, the bar graph legend, pie wedges and legend, and some control objects) cache their colors and therefore do not update when a custom color definition changes. To see the color change for these objects, restart Builder or reload the display.

*Deployment limitation:* Multiple applets running in the same VM share a single **Custom Color** tab.

## Setting options in the Apama tab group

The **Apama** tab allows you to define correlators and specify data management options. For information on the **Correlators** sub tab, see "Specifying correlators" on page 37.

For information on the Data sub tab, see "Specifying data sources" on page 37.

## Setting options in the SQL tab group

The SQL tab group has a single tab, **SQL**, which allows you to add or remove databases for use in Dashboard Builder and set a default database .

For more information on setting SQL options, see "Specifying application options" on page 251.

### Setting options in the XML tab group

The XML tab group has a single tab, The **XML tab**, which allows XML data files to be defined as data sources for use in Dashboard Builder.

These options are detailed in "Using XML Data" on page 235.

### Saving options

Clicking the **OK** or **Apply** button saves options for future use.

Dashboard Builder saves options to the file `OPTIONS.ini`. If Builder was started with a `--optionsFile` argument, the options are saved to the specified location. Otherwise, if the Builder current directory is your project's `dashboards` directory or the `dashboards` directory in your Apama installation's work directory, the options and are saved there. Otherwise, clicking **OK** brings up a dialog that allows you to specify the location to which to save the options.

Dashboard Builder saves custom colors to the file `COLORS.ini`. If the Builder current directory is your project's `dashboards` directory or the `dashboards` directory in your Apama installation's work directory, the colors (if modified) are saved there. Otherwise, clicking **OK** brings up a dialog that allows you to specify the location to which to save the custom colors.

If Builder was started without a `--optionsFile` argument, it uses the options file in its current directory, if present. Otherwise, it uses the options file in the `dashboards` directory in your Apama installation's work directory. In addition, Builder uses the colors file in its current directory, if present. Otherwise, it uses the colors file in the `dashboards` directory in your Apama installation's work directory, if present. Otherwise it uses the colors file in the `lib` directory of your Apama installation (which contains your Apama installation's initial set of custom colors).

## Command line options for the Dashboard Builder

The Dashboard Builder supports options that can be specified on the start-up command line to override the default values used by the builder. The executable for the Dashboard Builder is `dashboard_builder`, which can be found in the `bin` directory of the Apama installation.

#### Synopsis

To pass start-up options to the Dashboard Builder, run the following command:

```
dashboard_builder  [ options ] [ rtv-file-path ]
```

If you specify the full pathname of an `rtv` file, the builder will open it.

When you run this command with the `-h` option, the usage message for this command is shown.

### Options

The `dashboard_builder` executable takes the following options:

| Option | Description |
|---|---|
| `-B` *`logical-name:host:port`* \| `--namedServer` *`logical-name:host:port`* | Sets the host and port for a specified logical Data Server name. This overrides the host and port specified by the Dashboard Builder for the given server logical name. This option can occur multiple times in a single command. See "Working with multiple Data Servers" on page 72 for more information. |
| `-c` *`logical-name:host:port:bool`* \| `--correlator` *`logical-name:host:port:bool`* | Sets the correlator host and port for a specified logical correlator name. *`bool`* is one of `true` and `false`, and specifies whether to use the raw channel for communication. This overrides the host, port, and raw-channel setting specified by the Dashboard Builder for the given correlator logical name. See "Changing correlator definitions for deployment" on page 641. This option can occur multiple times in a single command. For example:<br>`-c default:localhost:15903:false`<br>`-c work1:somehost:19999:true`<br>These options set the host and port for the logical names `default` and `work1`. |
| `-D` *`directory`* \| `--dashboard` *`directory`* | Start with the dashboard found in the specified directory. |
| `-E` *`bool`* \| `--purgeOnEdit` *`bool`* | Specifies whether to purge all trend data when a scenario instance or DataView item is edited. *`bool`* is one of `true` and `false`. If this option is not specified, all trend data is purged when an instance is edited. In most cases, this is the desired mode of operation. |
| `-F` *`arg`* \| `--filterInstance` *`arg`* | Exclude instances which are not owned by the user. This option applies to all dashboard processes. Default is `false` for Dashboard Builder and `true` for the other dashboard processes.<br>Exception: when the Dashboard Viewer is connecting to a Dashboard Server, the default is `true` and cannot be overridden. |

| Option | Description |
|---|---|
| `-f` *file* \| `--logfile` *file* | Full pathname of the file in which to record logging. If this option is not specified, the options in the log4j properties file will be used. |
| `-G` *file* \| `--trendConfigFile` *file* | Trend configuration file for controlling trend-data caching. |
| `-h` \| `--help` | Emit usage information and then exit. |
| `-J` *file* \| `--jaasFile` *file* | Full pathname of the JAAS initialization file to be used by the Data Server. If not specified, the Data Server uses the file `JAAS.ini` in the `lib` directory of your Apama installation. |
| `-L` *file* \| `--xmlSource` *file* | XML data source file. If *file* contains static data, append `:0` to the file name. This signals Apama to read the file only once. |
| `-m` *mode* \| `--connectMode` *mode* | Correlator-connect mode. *mode* is one of `always` and `asNeeded`. If `always` is specified all correlators are connected to at startup. If `asNeeded` is specified, the Data Server connects to correlators as needed. If this option is not specified, the Data Server connects to correlators as needed. |
| `-N` *name* \| `--name` *name* | Set the component name for identification in the correlator. The default name is `Dashboard Builder:` *username*. |
| `-n` \| `--noSplash` | Do not display splash screen in startup. |
| `-O` *file* \| `--optionsFile` *file* | Use the specified `OPTIONS.ini` file at startup. |
| `-P` *n* \| `--maxPrecision` *n* | Maximum number of decimal places to use in numerical values displayed by dashboards. Specify values between 0 and 10, or -1 to disable truncation of decimal places. A typical value for *n* is 2 or 4, which eliminates long floating point values (for example, 2.2584435234). Truncation is disabled by default. |

| Option | Description |
|---|---|
| `-q` *options* \| `--sql` *options* | Configures SQL Data Source access. *options* has the following form:<br>`[retry:`*ms* `\| fail:`*n* `\| db:`*name* `\| noinfo \| nopererr \| quote]`<br>`retry`: Specify the interval (in milliseconds) to retry connecting to a database after an attempt to connect fails. Default is `-1`, which disables this feature.<br>`fail`: Specify the number of consecutive failed SQL queries after which to close this database connection and attempt to reconnect. Default is `-1`, which disables this feature.<br>`db`: Specify the logical name of the database as specified in the builder's SQL options.<br>`nopererr`: SQL errors with the word "permission" in them will not be printed to the console. This is helpful if you have selected the **Use Client Credentials** option for a database. In this case, if your login does not allow access for some data in their display, you will not see any errors.<br>`quote`: Encloses all table and column names specified in the Attach to SQL Data dialog in quotes when an SQL query is run. This is useful when attaching to databases that support quoted case-sensitive table and column names. If a case-sensitive table or column name is used in the **Filter** field, or you are entering an advanced query in the **SQL Query** field, they must be entered in quotes, even if the `-sqlquote` option is specified. |
| `-R` *bool* \| `--purgeOnRemove` *bool* | Specifies whether to purge all scenario or DataView data when an instance or item is removed. *bool* is one of `true` and `false`. If this option is not specified, all scenario and DataView data is purged when an instance or item is removed. |
| `-S` *variable*`:`*value* \| `--sub` *variable*`:`*value* | Specifies a value to substitute for a given dashboard variable. This can be used to parameterize a dashboard at startup. This option can occur multiple times in a single command. For example:<br>`-S $foo:hello -S $bar:can't -S $tom:"my oh my" -S $jerry:"\"yikes\""` |

| Option | Description |
| --- | --- |
| | If the value contains a space, enclose the value in double quotes. If the value contains a double quote, you must escape it by using a backslash character (\). |
| `-T` *`depth`* \| `--maxTrend` *`depth`* | Maximum depth for trend data, that is, the maximum number of events in trend tables. If this option is not specified, the maximum trend depth is 1000. Note that the higher you set this value, the more memory the Data Server requires, and the more time it requires in order to display trend and stock charts. |
| `-t` *`value`* \| `--title` *`value`* | Text for the title bar of the Dashboard Builder main window. |
| `-u` *`rate`* \| `--updateRate` *`rate`* | Data update rate in milliseconds. This is the rate at which the Data Server pushes new data to deployed dashboards in order to inform them of new events received from the correlator. *`rate`* should be no lower than 250. If the Dashboard Viewer is utilizing too much CPU, you can lower the update rate by specifying a higher value. If this option is not specified, an update rate of 500 milliseconds is used. |
| `-V` \| `--version` | Emit program name and version number and then exit. |
| `-v` *`level`* \| `--loglevel` *`level`* | Logging verbosity. *`level`* is one of `FATAL`, `ERROR`, `WARN`, `INFO`, `DEBUG`, and `TRACE`. If this option is not specified, the options in the log4j properties file will be used. |
| `-w` *`bool`* \| `--disconnectWarning` *`bool`* | By default, the Dashboard Builder will display a warning dialog when the connection to a correlator is lost. Specify `false` to disable the display of this dialog. |
| `-X` *`file`* \| `--extensionFile` *`file`* | Full pathname of the JAAS initialization file to be used by the Data Server. If not specified, the Data Server uses the file `EXTENSIONS.ini` in the `lib` directory of your Apama installation. |

| Option | Description |
|---|---|
| `-x table-name:key-list` \| `--queryIndex table-name:key-list` | Add an index for the specified SQL-based instance table with the specified compound key. *table-name* is the name of a scenario or DataView. *key-list* is a comma-separated list of variable names or field names. If the specified scenario or DataView exists in multiple correlators that are connected to the Dashboard Server, the index is added to each corresponding data table. Example:<br>`--queryIndex Products_Table:prod_id,vend_id`<br>You can only add one index per table, but you can specify this option multiple times in a single command line in order to index multiple tables. |
| `-Y` \| `--enhancedQuery` | Make SQL-based instance tables available as data tables for visualization attachments. See "Attaching Dashboards to Correlator Data" on page 55. |
| `-z zone` \| `--timezone zone` | Default time zone for interpreting and displaying dates. *zone* is either a Java timezone ID or a custom ID such as `GMT-8:00`. Unrecognized IDs are treated as GMT. See "Timezone ID Values" on page 715 for the complete listing of permissible values for *zone*. |
| `--exclusionFilter val` | Set scenario exclusion filters. This option can occur multiple times in a single command. |
| `--inclusionFilter val` | Set scenario inclusion filters. This option can occur multiple times in a single command. |

## Restrictions

This section lists the known restrictions of using Dashboards.

- The substitution names should not contain any of the following characters:

  Colon (:), pipe (|), period (.), comma (,), semi-colon (;), equals (=), brackets (< >, ( ), { }, [ ]), quotation marks (' "), ampersand (&), slashes (/ \)

# 3 Attaching Dashboards to Correlator Data


- Dashboard data tables ..... 56
- Scenario instance and DataView item ownership ..... 63
- Creating a data attachment ..... 64
- Using table objects ..... 76
- Using pie and bar charts ..... 88
- Using trend charts ..... 91
- Using stock charts ..... 111
- Localizing dashboard labels ..... 129
- Localizing dashboard messages ..... 133

A key feature of Dashboard Builder is the ability to attach visualization objects such as tables and charts to live correlator data. This feature enables dashboards to display correlator activity in real time.

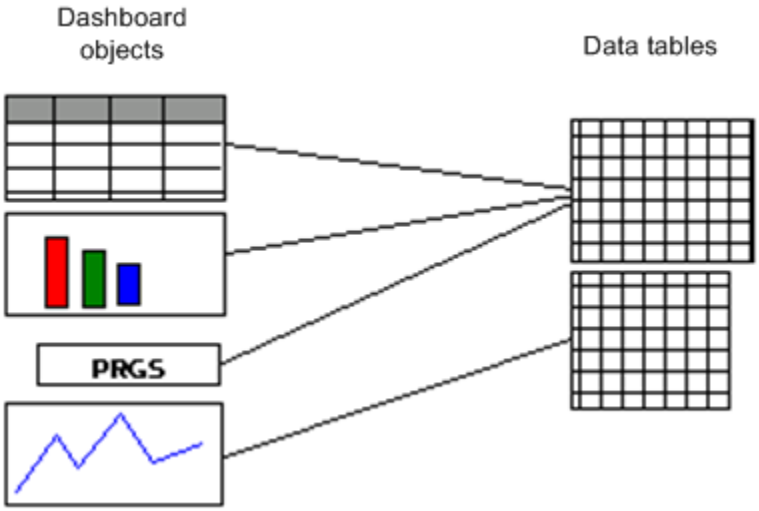
You can attach visualization objects to two kinds of correlator data: scenario data and DataView data. Scenarios and DataViews are described in *Introduction to Apama*.

This chapter describes the data that is available for attachment, and it describes the most common objects that can be attached to the data. The examples focus on a sample trading scenario (see "Using the tutorial application" on page 27). Dashboard Builder provides many objects that can be included in a dashboard. This chapter does not detail each one for both scenario and DataView data, but upon completion of this chapter you should be comfortable with using any Dashboard Builder object with a scenario or DataView.

## Dashboard data tables

To create dashboards, you should have an understanding of how Apama manages correlator data and makes it available for attachment to object properties.

Apama makes scenario and DataView data available to dashboards as tabular data. Multiple data tables may be necessary for a dashboard. Any data table may have multiple objects in the dashboard attached to it. The relationship between dashboard objects and data tables is illustrated in the following diagram.




When a scenario variable or DataView field changes, the correlator generates an update event with details of the change. When this event is received by a dashboard, the dashboard updates one or more data tables and the changes are reflected in all attached objects.

Different data tables are used for each scenario or DataView. Data tables are not created until the first attachment requiring the data table is made. In the Dashboard Builder this happens when the attachment is defined. For a deployed dashboard, this happens when the dashboard is launched or loaded.

Once created, a data table exists for the life of the Builder process or deployed-dashboard session, although it may be purged of data if the corresponding scenario or DataView definition is deleted from the correlator or if the scenario instance or DataView item is deleted.

Apama filters the scenario instances or DataView items a user can see. Only those instances that the user is authorized for will be added to the user's data tables. By default, these are the scenario instances or DataView items that the user created. See Administering Dashboard Security in *Deploying Apama Applications* for more information on dashboard authorization.

## Scenario instance table

A scenario instance data table contains the current values of all variables for all instances of a single scenario definition. A separate instance table exists for each scenario. Within a scenario instance data table, a row exists for each instance of the scenario. The columns of the table correspond to the input and output variables of the scenario.

The table below illustrates the contents of a scenario instance table.

| Instrument | Price | Velocity | Shares | Position |
|---|---|---|---|---|
| APMA | 28.5 | 0.0125 | 10000 | 285000 |
| ORCL | 12.3 | -0.0173 | 12500 | 153750 |
| MSFT | 26.4 | 0.0 | 8000 | 211200 |

Here there are three instances of the scenario; each row corresponds to one instance. The scenario has five variables; each column corresponds to one scenario variable.

Apama adds several additional columns to each scenario instance table that contain information not available as scenario variables. These additional columns include the following:

- `apama.instanceId`: The value is an id string which can be used to uniquely identify the scenario instances. This id string is used when performing drilldowns or operations on a scenario instance
- `apama.instanceStatus`: The value is a string which identifies the status of the scenario instance. Possible values are:
  - `RUNNING`: The instance is running
  - `ENDED`: The instance terminated normally
  - `FAILED`: The instance terminated abnormally
- `apama.owner`: The value is the owner of the scenario instance, typically the ID of the user that created it.

- `apama.substitutions` — Do not use this column. It will be removed in a future release.
- `apama.timestamp`: The value is a UTC timestamp which indicates the time the last Update event was received for the scenario instance.

The scenario instance table would be as follows.

| Instrument | Price | Velocity | Shares | Position | apama. instanceId | apama. instanceStatus | apama. timestamp |
|---|---|---|---|---|---|---|---|
| APMA | 28.5 | 0.0125 | 10000 | 285000 | *ID* | RUNNING | *Timestamp* |
| ORCL | 12.3 | -0.0173 | 12500 | 153750 | *ID* | ENDED | *Timestamp* |
| MSFT | 26.4 | 0.0 | 8000 | 211200 | *ID* | RUNNING | *Timestamp* |

Scenario instance tables will likely be used by any dashboard you create. They are the only data table which contains the values of the scenario input variables.

## Scenario trend table

A scenario trend table contains the values of variables of a single scenario instance. A separate data table is used for each instance of a scenario. Each row in the table contains the value of the variables as reported in an Update event. Each row also contains a timestamp indicating when the Update occurred.

The following diagram illustrates the contents of a scenario trend table.

| apama. timestamp | Price | Velocity | Shares | Position |
|---|---|---|---|---|
| T0 | 28.5 | 0.0125 | 10000 | 285000 |
| T1 | 28.5 | 0.0 | 9900 | 282150 |
| T2 | 28.4 | -0.125 | 9900 | 281160 |

Here the table contains the values of three Update events occurring at times T0, T1, and T2.

Trend tables are limited in size; by default they will hold 1000 rows. The maximum row count is a configurable option. When a data table is full each new Update event will result in the oldest row being removed and a new row being added.

Trend tables are for use with trend and stock charts where you want to graph the changes of a variable value over time.

## Scenario OHLC table

A scenario OHLC table contains Open, High, Low, and Close values for a scenario variable as calculated for a specified time interval. As a dashboard or dashboard server receives update events for a scenario instance it will calculate the Open, High, Low, and Close values for the variable and add a row to an OHLC table at each time interval. The calculated values added will be for the preceding time interval.

OHLC tables allow dashboards to automatically create data suitable for display in a Candlestick or OHLC chart for any scenario variable and time interval. When you create an attachment to an OHLC table you specify the variable and time interval desired. An example would be selecting a Price variable and a time interval of 5 seconds.

A separate OHLC table is used for each scenario instance and each variable and interval pair. If for the `Price` variable you wanted OHLC data at both 5 and 30 second intervals; two OHLC tables would be created for each instance of the scenario.

The following table illustrates the contents of a scenario OHLC table:

| apama.timestamp | Open | High | Low | Close |
|---|---|---|---|---|
| T0 | 28.5 | 29.1 | 28.3 | 28.4 |
| T0 + interval | 28.4 | 28.6 | 28.4 | 28.5 |
| T0 + (interval* 2) | 28.5 | 29.0 | 28.3 | 28.7 |

Each row in an OHLC table contains a timestamp indicating when the row was added to the table. This is the end time of each interval.

OHLC tables are limited in size; by default they will hold 1000 rows. The maximum row count is a configurable option. When a data table is full each new Update event will result in the oldest row being removed and a new row being added.

OHLC tables are for use with stock charts to display candlestick or OHLC graphs of a scenario variable over time. The benefit of OHLC tables is that they allow you to use the stock chart without modifying your scenario to generate OHLC values; Apama can do it for you.

## Correlator status table

A single correlator status table contains status information about each correlator being used by a dashboard. It is useful when you want to display status information about correlator connections in a dashboard.

The following table illustrates the contents of the correlator status table.

| logical name | host | port | status |
|---|---|---|---|
| default | localhost | 15903 | connected |
| production | linux23 | 15903 | connected |

Here two correlators are in use and each is connected.

## Data Server status table

A single data server status table contains status information for the Data Server being used by a dashboard. It is useful when you want to display status information about the Data Server connection in a dashboard.

The table below illustrates the contents of the Data Server status table.

| Name | Status | ConnectionString | ReceiveCount | ReceiveTime | Config |
|---|---|---|---|---|---|
| __default | no connection | localhost:3278 | 0 | Dec 31, 1969 6:00... | <Data Server version> |

(This type of table differs from the others in that it cannot be attached to a property with the **Attach to Apama** dialog—see "Creating a data attachment" on page 64. To attach a property to a Data Server status table, attach the property to function data—see "Using Dashboard Functions" on page 135—and specify a function of type **Get Data Server Connection Status**.)

## Scenario constraint table

A scenario constraint data table contains the metadata for all the variables of a specified scenario. A separate constraint table exists for each scenario. The table has a row for each variable and a column for each kind of metadata.

The table below illustrates the contents of a scenario constraint table.

| Parameter | Case | Choices | Constant | Default | Maximum | Minimum | Mutable | Trim | Type | Unique |
|---|---|---|---|---|---|---|---|---|---|---|
| Instrument | mixed | null | 0 | | null | null | 1 | 1 | string | 0 |
| Price | null | null | 0 | | null | null | 1 | 1 | float | 0 |
| Velocity | null | null | 0 | | null | null | 1 | 1 | float | 0 |

| Parameter | Case | Choices | Constant | Default | Maximum | Minimum | Mutable | Trim | Type | Unique |
|---|---|---|---|---|---|---|---|---|---|---|
| Shares | null | null | 0 | | null | null | 1 | 1 | integer | 0 |
| Position | null | null | 0 | | null | null | 1 | 1 | integer | 0 |

See "Attaching to constraint data" on page 67 for more information on using constraint tables.

## DataView item table

A DataView item data table is similar to a scenario instance table (see "Scenario instance table" on page 57). It contains the current values of all fields for all items of a single DataView definition. A separate item table exists for each DataView definition. Within a DataView item table, a row exists for each item associated with a specified DataView definition. The columns of the table correspond to the fields of the DataView.

## DataView trend table

A DataView trend data table is similar to a scenario trend table (see "Scenario trend table" on page 58). It contains the values of the fields of a single DataView item. A separate data table is used for each item associated with a DataView definition. Each row in the table contains the value of the fields as reported in a DataView-item update event. Each row also contains a timestamp indicating when the update occurred.

## DataView OHLC table

A DataView OHLC table is similar to a scenario OHLC table (see "Scenario OHLC table" on page 59). It contains Open, High, Low, and Close values for a DataView item field as calculated for a specified time interval. As a dashboard or dashboard server receives update events for a DataView item it will calculate the Open, High, Low, and Close values for the field and add a row to an OHLC table at each time interval. The calculated values added will be for the preceding time interval.

OHLC tables allow dashboards to automatically create data suitable for display in a Candlestick or OHLC chart for any DataView-item field and time interval. When you create an attachment to an OHLC table you specify the field and time interval desired. An example would be selecting a Price field and a time interval of 5 seconds.

A separate OHLC table is used for each DataView item and each field and interval pair. If for the `Price` field you wanted OHLC data at both 5 and 30 second intervals; two OHLC tables would be created for each DataView item.

## SQL-based instance table

An SQL-based data table is a special data table designed to ease implementation of complex filtering and improve performance for dashboards that must handle a large number of scenario instances or DataView items. It is similar to a scenario instance table (see "Scenario instance table" on page 57) and a DataView item table (see

"DataView item table" on page 61). It contains the current values of all input and output variables for all instances of a single scenario, or the current values of all fields for all items of a single DataView definition.

A separate table exists for each scenario or DataView definition. Within a table, a row exists for each instance of the scenario or item of the DataView definition. The columns of the table correspond to the variables of the scenario or fields of the DataView.

See "Using SQL-based instance tables" on page 70 for more information on using SQL-based instance tables.

When you specify a data attachment, this kind of table is available only if you started Builder with the `-Y` or `--enhancedQuery` command line option.

**Important:** When SQL-based data tables are in use for deployed dashboards, authorization for scenario instances and DataView items does not use scenario authorities (see "Administering Dashboard Security" on page 677). By default, all users have access to all instances or items. Authorization must be built into attachment queries. See "Using SQL-based instance tables" on page 70 for more information.

## Definition Extra Params table

A definition extra params table contains the metadata for the `extraParams` member of the selected `Definition` (Scenario, DataView or Query). The table has two static columns: `key` and `value`. An entry in `extraParams` is considered a metadata entry if its `key` name starts with the `Metadata:` prefix. Each metadata entry in the `extraParams` member will appear as a row in this table.

For example, a DataView definition may have extraParam like this:

```
com.apama.dataview.DataViewAddDefinition
  add := new com.apama.dataview.DataViewAddDefinition;
add.dvName := "RecipeDV";
add.dvDisplayName := "Recipe";
add.fieldNames := ["name","ingredients","category","difficulty"];
add.fieldTypes := ["string","string","string","integer"];
add.keyFields  := ["name"];
add.extraParams := {
   "Metadata:author":"John Doe",
   "Metadata:copyrightDate":"October 15, 2015",
   "Metadata:contact":"jdoe@kitchen.com",
   "phone":"234-123-9988",
   "age":"30"};
```

In the above example, only the first 3 `extraParams` entries are metadata entries. Therefore, the Definition extra params table will show:

| key | value |
|---|---|
| author | `John Doe` |
| copyrightDate | `October 15, 2015` |

| key | value |
|---|---|
| contact | `jdoe@kitchen.com` |

### Setting data options

Dashboard Builder provides several options for managing the data stored in data tables.

**To set data options**

1. Select **Options** item in the **Tools** menu.

   The Application Options dialog appears.

2. Select the **Apama** tab and the **Data** sub tab to see the data options.
3. Check the **Purge instance on edit** check box to purge all trend and OHLC data for a scenario instance or DataView item whenever an input variable or field is modified. When an input variable of a scenario or field of a DataView item is modified, it may invalidate all previous trend and OHLC data.
4. Check the **Purge scenario data on remove** to purge all data for a scenario or DataView when it is removed from a correlator.
5. Check the **Maximum decimal precision** and specify a maximum number of decimal places to be displayed for any numeric data in a dashboard.
6. Check the **Maximum rows per trend table** to set the maximum number of rows for each trend and OHLC table. The higher the value, the more data that will be available for charting and the greater the memory utilization.

## Scenario instance and DataView item ownership

Scenario instances and DataView items in a correlator include an attribute identifying the owner of the instance. When a scenario instance is created through a dashboard, it provides the current user ID as the owner of the instance.

By default, you are only allowed to see and operate on those scenario instances and DataView items that you own, that is, the current user ID must match the `apama.owner` attribute of the instance or item. There are two exceptions to this default:

- If the owner is specified as "`*`", all users have access by default.
- SQL query attachments provide access for all users to all instances and items. See "Using SQL-based instance tables" on page 70 for more information on SQL query attachments.

## Creating a data attachment

Attachments can be used to provide data for a chart or table. They can also be used to set other properties of objects such as labels, colors, and thresholds. Any non-static object property can be attached to Apama data.

The value of a property, for a given visualization object, can be a single numeric or string value, a sequence of values, or a table of values. The value of an object property can specify a set of characteristics of the object, such as the following:

- Numerical contents of all the cells in a table
- Height and label of all the bars in a bar graph
- X coordinate and Y coordinate of all the plotted points in an XY Graph

For example, the value of the **valueTable** property for a basic bar graph is a table that has one row for each bar in the graph. The first column in each row provides the label for the corresponding bar, and the second column in the row provides the height of the corresponding bar.

### Using the Attach to Apama dialog

**To attach an object property to Apama data**

1. Right-click the property in the property panel.

   A popup menu appears.

2. In the displayed popup menu pick **Attach to Data > Apama**.

   This displays the **Attach to Apama** dialog.

   This dialog allows you to specify the portion of a data table that is to be used as the object property's value. This portion is itself a table consisting of some or all of the rows and columns of the original data table. The dialog, in effect, allows you to specify a query against a specified data table. At any given time, the result of this query serves as the value of the object property being attached.

3. In the **Attach to** field select the type of Apama data table needed:
   - scenario instance
   - scenario trend
   - scenario OHLC
   - scenario constraint
   - correlator status
   - DataView item
   - DataView trend

- DataView OHLC
- DataView constraint.

To attach a property to an SQL-based data table, see "Using SQL-based instance tables" on page 70.

To attach a property to a Data Server status table, attach the property to function data—see "Using Dashboard Functions" on page 135—and specify a function of type **Get Data Server Connection Status**.

4. In the **For** field, if the **Attach to** field specifies a scenario or Data View trend or OHLC table, select **History and new events**, **New events only**, or **History only**. This specifies whether to attach new or historical data to this property.
5. In the **Correlator** field enter the correlator where the scenario or DataView is loaded. This field is disabled if the **Attach to** field specifies a correlator status table.
6. In the **Scenario** or **DataView** field, enter the scenario or DataView definition to attach to. This field is disabled if the **Attach to** field specifies a correlator status table.
7. In the **Timestamp variable** field, for trend table and OHLC table attachments, identify a scenario variable, DataView field, or `apama.timestamp` to use as the timestamp for rows in the data table.
8. In the **Display variables** field enter the data table columns (which are scenario variables or DataView fields) to include in the portion of the table to be used as object property value.
9. Check the **Filter** check box to enable the filter fields (listed below). Filters allow you to specify the data table rows to include in the value of the attached property. You do this by specifying a condition that must be satisfied by a data table row in order for it to be included. The condition specifies a data table column, a value, and a comparison relation (for example, **equals**, **less than**, or **member of**). The condition is satisfied by a given row if and only if the value of the specified column for the row bears the specified relation to the specified value. Enter the filter field values:
   a. **By variable** — Specifies the data table column (which is a scenario variable or DataView field) to filter against.
   b. Comparison operator field — Specifies one of the following comparisons. To compare numeric or text values, use **equals**, **not equals**, **greater than**, **greater than or equals**, **less than**, or **less than or equals**. Use **member of** to compare a column value with a list of numeric or text values. Use **starts with**, **ends with**, or **contains** to compare text values only.
   c. **Value** — Specifies the value to compare with values of the specified column. For **Member of** comparisons, specify a single value or a semi-colon-separated list of values. Do not use spaces. A single value is considered to be a list with a single member. Escape quotes in values (that use `\'` instead of `'`).

      See "Filtering data" on page 66 for more information.

10. **Using time interval** — For OHLC table attachments specifies the time interval to be used in calculating OHLC values.
11. **Data Server** — For advanced users, specifies the logical name of the Data Server that you want to serve the data associated with this attachment. You define Data Server logical names with the **Application Options** dialog (select **Tools Options** ). See "Working with multiple Data Servers" on page 72 for more information.

    In this documentation, some of the **Attach to Apama** dialogs are shown without the **Data Server** field, which has been added in a later release.

## Selecting display variables or fields

Individual display variables or fields can be selected directly in the Attach to Apama dialog.

**To select multiple display variables or fields**

1. Click "..." next to the **Display variables** field.

   This displays the Select Columns dialog.
2. Select and order multiple display variables or fields using the buttons provided.

## Displaying attached data

**To view the data that is currently attached to a given property**

1. Right-click the property name.

   A popup menu appears.
2. Select **Display Data** from the popup menu.

   A dialog appears that contains a table and the following checkboxes:

   - **Show Column Types**: Provides the option of displaying data-table column types.
   - **Insert New Rows**: Controls whether new data is added to the table as new rows instead of replacing the old rows.
   - **Scroll Columns**: Controls whether a scrollbar is provided when needed to prevent truncation of column contents.

## Filtering data

The **Attach to Apama** dialog allows you to define a filter, which specifies a condition on rows of a data table. Only rows that satisfy the condition are included in the table that serves as the value of the attached property. See "Using the Attach to Apama dialog" on page 64 for details on specifying filter conditions.

Filters are used frequently in dashboards. Most frequently they are used to select a single scenario instance or DataView item for which dashboard objects are to display

**Note:** When you create an attachment to an instance or item table, constraint table, or correlator status table, the filter identifies the rows in the table you want to use. When you create an attachment to a trend or OHLC table, the filter identifies the table to use.

## Attaching to constraint data

When you attach a property to data from a constraint table, you use the **Attach to Apama** dialog to specify a single cell of the constraint table (the dialog requires you to specify a single column for **Display Variables** and to filter on the value of the **Parameter** column). The contents of this cell is used as the property's value. Use this kind of attachment to set constraints on controls, such as the maximum value on a slider.

## About timestamps

When creating a stock or trend chart data attachment, you must identify the variable or field to use as the timestamp. You can use either a scenario variable, DataView field, or `apama.timestamp`. When a variable or field changes, the correlator generates an Update event with the new value. The timestamp in the Update event will be used by the dashboard as the time that the change occurred and used to chart the value.

The default timestamp is `apama.timestamp`. It corresponds to the timestamp the correlator adds to an Update event when the event is generated. This timestamp is suitable in most cases and is always available.


If you want greater control over the value of timestamps, specify a scenario variable or DataView field as the timestamp. Within your scenario or DataView you will need to set the value of the timestamp variable or field when changing the value of any other variable or field. Do this if you want use timestamps from an external event feed such as market data.


Here the scenario variable named `timestamp` is being used.

Only number variables can be used as timestamps. Timestamps need to be in UTC format where the value represents the number of seconds since the epoch, January 1, 1970. The MonitorScript `TimeFormatPlugin` can be used to convert string values to UTC format.

## Using dashboard variables in attachments

The value of all fields in the **Attach to Apama** dialog, other than **Attach to** and **For**, can be set to dashboard *substitution variables*. This allows you to dynamically configure an attachment when a dashboard is displayed. For example you could set the **Display variables** field to the substitution variable `$displayVariables` (where `$displayVariables` value equals a semicolon separated list of scenario variables).

**To create a substitution variable**

1. Select **Tools | Variables** to display the **Variables** panel (if the panel is not showing).
2. In the **Name** field enter a name that starts with "`$`". Names of substitution variables start with "`$`" by convention. Names of variables that are not substitution variables (see below) do not start with "`$`".
3. In the **Initial Value**, optionally supply an initial value.
4. Check the **Use as substitution** checkbox.
5. In the **Data Type** field, ensure that this set to **Scalar**, the default.

The Initial Value field allows static specification of substitution values at development time. You can also allow dashboard users to set the value of a given substitution at runtime by attaching the `varToSet` property of a control object (such as a text field) to the given substitution.

Dashboard Builder provides a number of predefined substitutions—see "About drilldown and $instanceId" on page 68 and "About other predefined substitution variables" on page 69.

Dashboard variables in attachments only take effect when the dashboard is displayed. Subsequent changes to the variable will not change the attachment unless the dashboard is redisplayed.

#### *About non-substitution variables*

In addition to using dashboard variables as field values in the **Attach to Apama** dialog, you can specify a dashboard variable as the value of an object property. If you use a variable in this way, you can increase dashboard efficiency by clearing the selection **Use as substitution** field for the variable in the **Variables** panel, provided you do *not* use the variable in any of the following:

- **Attach to Apama** field
- **Define Apama Command** field (see "Defining commands" on page 150)
- `-s` command line option

Substitution variables must have a scalar value, but non-substitution variables can have tabular values if you set the **Data Type** to **Table**.

Uncheck the **Public** checkbox only if you do *not* want to expose the variable as a property in a Composite object—see "Using Composite objects" on page 174.

## About drilldown and $instanceId

When you create a dashboard with Dashboard Builder, you will frequently need to pass context information that identifies a scenario instance or DataView item to display or operate on. Consider, for example, a dashboard with a table containing one row for each instance of a given scenario. In order to display detailed information about a scenario instance when the end user selects its corresponding row in the table, you need to pass

the identity of the selected instance to the visualization objects that will display the details.

You can pass such information from one object to another by doing both the following:

- Specify that a substitution variable be set to a specified value in response to a specified end-user action on one object.
- Use that substitution variable in the data attachment for the other object.

In many cases you can simplify this procedure by using the pre-defined substitution variable `$instanceId`. This variable is automatically set to the value of `apama.instanceId` (see "Scenario instance table" on page 57) for the table row that is currently selected (for tables attached to a scenario instance table). If multiple rows are selected, `$instanceId` is set to multiple values.

For more information and examples, see "Performing drilldowns on tables" on page 79 and "Specifying drill-down column substitutions" on page 82.

**Note:** In cases where the end user can select rows of multiple tables at once, you must use user-defined variables instead of `$instanceId` to pass the required information. If rows from multiple tables are selected, `$instanceId` is set according to only one of the tables.

You will find yourself using `$instanceId` frequently in attachment filters and scenario operations. You will see many uses of `$instanceId` in subsequent sections of this guide.

## About other predefined substitution variables

In addition to `$instanceId` (see "About drilldown and $instanceId" on page 68), Dashboard Builder defines the following substitution variables:

- `$apama_lang`: by default, this variable is set to what Java reports as the locale in the `Locale` object as derived from the host system's locale. You can allow end users to set this to their required locale, and use it to localize dashboard labels. See "Localizing dashboard labels" on page 129.
- `$apama_roles`: `Principles`: returned by the login module.
- `$apama_server_host`: hostname of the machine running the Data Server or Display Server; empty for Builder and Viewer with a direct connection to a Correlator.
- `$apama_server_port`: port used by the Data Server or Display Server on the host machine; empty for Builder and Viewer with a direct connection to a Correlator.
- `$apama_timestamp`: by default, this variable is set to the value of `apama.timestamp` of the scenario instance that is currently selected. See "About timestamps" on page 67.
- `$apama_user`: current user, set at login.
- `$celldata`: by default, this variable is set to the value of the cell that is currently selected.

- `$colName`: by default, this variable is set to the name of the column of the currently-selected cell.

## Using SQL-based instance tables

SQL-based instance tables support the use of an SQL query for the specification of a data attachment. (See "SQL-based instance table" on page 61 for a description of the contents of this type of table.) By using these tables, you can simplify your implementation of complex filtering, and improve performance for dashboards that must handle a large number of scenario instances or DataView items. In particular, SQL-based instance tables have the following potential advantages over other types of data tables (which require you to use the standard fields of the **Attach to Apama** dialog):

- Filtering is optimizable. You can specify indexes which Apama can use to join data tables and filter data attachments more efficiently. This can dramatically improve performance, particularly for large data tables (that is, tables with thousands of rows or more).
- A single attachment specification can refer to multiple tables, including tables from multiple correlators. This can simplify implementation, which would otherwise require attaching properties to dashboard functions whose arguments are attached to data tables.
- Arbitrarily complex filtering and data aggregation is supported, since any read-only SQL `select` statement can be used. This can simplify implementation, which would otherwise require complex chains of dashboard functions.

**Important:** When SQL-based data tables are in use for deployed dashboards, authorization for scenario instances and DataView items does not use scenario authorities (see "Administering authentication" on page 678). By default, all users have access to all instances or items. Authorization must be built into attachment queries.

**To attach an object property to Apama data by using an SQL-based instance table**

1. Ensure that Builder has been started with the -Y or `--enhancedQuery` command line option.
2. Right-click the property in the property panel.

   A popup menu appears.
3. In the displayed popup menu pick **Attach to Data > Apama**.

   The **Attach to Apama** dialog appears.
4. In the **Attach to** field select **Instance table query**.

   This changes the **Attach to Apama** dialog, so that there is a single remaining field, **SQL Statement**.
5. Enter an SQL query into the text box.

Any read-only `select` statement is allowed, with the following restrictions and modifications:

- You must designate tables with table names of the form correlator-name.scenario-or-data-view-ID.
- You can designate values with predefined or user-defined dashboard substitution variables (for example, `$apama_user` or `$instanceId`).
- You must enclose table names and column names in quotes.
- You must enclose strings in single quotes.

As you construct your query, you can right-click to get suggestions for table names, column names, or substitution variables.

**Note:** Errors in the SQL query are logged in the dashboard log file.

Following is an example of a query that you can use to specify a data attachment. It specifies a three-way join, that is, a join involving three different data tables:

```
SELECT "prod_name", "vend_name", "prod_price", "quantity"
  FROM "Correlator2.DV_OrderItems_Table", "Correlator1.DV_Products_Table",
    "Correlator1.Scenario_Vendors_Table"
  WHERE "Correlator1.DV_Products_Table"."vend_id" =
      "Correlator1.Scenario_Vendors_Table"."vend_id"
    AND "Correlator2.DV_OrderItems_Table"."prod_id" =
      "Correlator1.DV_Products_Table"."prod_id"
    AND "Correlator2.DV_OrderItems_Table"."order_num" = 20007
```

Below is a query that filters out instances that are not owned by the current dashboard user. The example assumes that there is a scenario variable or DataView field, `owner`, whose value is the instance owner.

```
SELECT "prod_id", "prod_price"
  FROM "Correlator1.Scenario_Vendors_Table"
  WHERE "Correlator1.Scenario_Vendors_Table"."owner" = '$apama_user'
```

To specify indexes into an SQL-based data table, use the `--queryIndex` option on the command line when you do any of the following:

- Start the Data Server or Display Server
- Start the Dashboard Builder with a direct connection to the correlator
- Start the Dashboard Viewer with a direct connection to the correlator

This option has the form

```
--queryIndex table-name:key-list
```

*`table-name`* is the name of a scenario or DataView. *`key-list`* is a comma-separated list of variable names or field names. Here is an example:

```
--queryIndex DV_Products_Table:prod_id,vend_id
```

You can only add one index per table, but you can specify this option multiple times in a single command line in order to index multiple tables. Deployed dashboards that use SQL-based instance tables must be connected to a Data Server or Display Server that is started with the `-Y` or `--enhancedQuery` command line option. For deployed

dashboards that use Viewer connected directly to a correlator, Viewer must be started with the `-Y` or `--enhancedQuery` command line option.

## Working with multiple Data Servers

Deployed dashboards have a unique associated default Data Server or Display Server. For Web-based deployments, this default is specified in the Startup and Server section of the Deployment Configuration Editor. For Viewer deployments, it is specified upon Viewer startup. By default, the data-handling involved in attachments and commands is handled by the default server, but advanced users can associate non-default Data Servers with specific attachments and commands. This provides additional scalability by allowing loads to be distributed among multiple servers. This is particularly useful for Display Server deployments. By deploying one or more Data Servers behind a Display Server, the labor of display building can be separated from the labor of data handling. The Display Server can be dedicated to building displays, while the overhead of data handling is offloaded to Data Servers.

Apama supports the following multiserver configurations:

- Builder with multiple Data Servers. See "Builder with multiple Data Servers" on page 73.
- Viewer with multiple Data Servers. See "Viewer with multiple Data Servers" on page 73.
- Display Server (thin client) deployment with multiple Data Servers. See "Display Server deployments with multiple Data Servers" on page 74.
- Applet or WebStart deployment with multiple Data Servers. See "Applet and WebStart deployments with multiple Data Servers" on page 75.

The **Attach to Apama** and **Define ... Command** dialogs (except **Define System Command**) include a **Data Server** field that can be set to a Data Server's logical name. To associate a logical name with the Data Server at a given host and port, use the **Data Server** tab in the **General** tab group of the **Application Options** dialog (select **Tools Options** in Builder).

For Display Server (thin client) deployments, you must use the option `--namedServerMode` whenever you start named Data Servers. See "Display Server deployments with multiple Data Servers" on page 74.

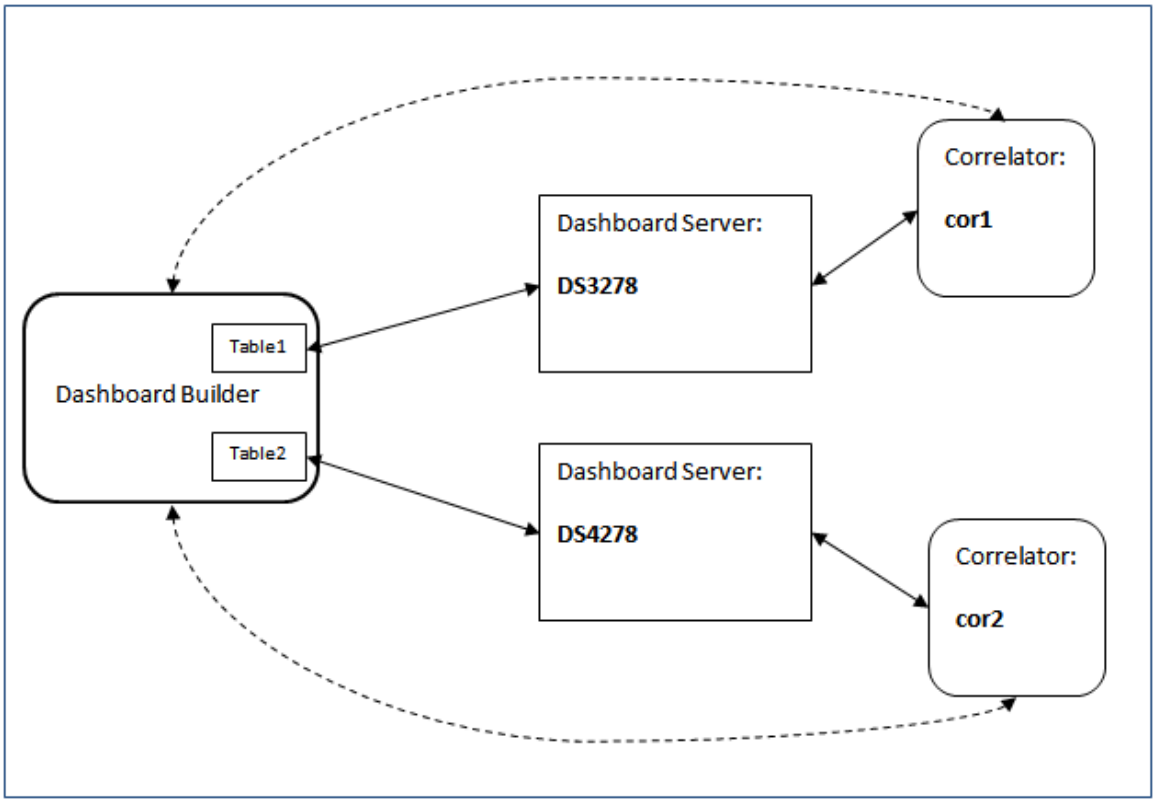
The logical Data Server names specified in the Builder's **Application Options** dialog are recorded in the file `OPTIONS.ini`, and the deployment wizard incorporates this information into deployments. You can override these logical name definitions with the `--namedServer` *name*`:`*host*`:`*port* option to the Builder, Viewer, Data Server or Display Server executable. Below is an example. This is a sequence of command line options which should appear on a single line as part of the command to start the executable:

```
--namedServer Server1:ProductionHost_A:3278 --
namedServer Server2:ProductionHost_B:4278 --namedServer
Server3:ProductionHost_C:5278
```

Here `Server1`, `Server2` and `Server3` are the server logical names.

### *Builder with multiple Data Servers*

Builder maintains connections with the Data Servers named in attachments and commands. Note that it connects directly to the correlator (dotted lines in the figure below) in order to populate dialogs with metadata. In this illustration, correlator event data is handled by the Data Servers.



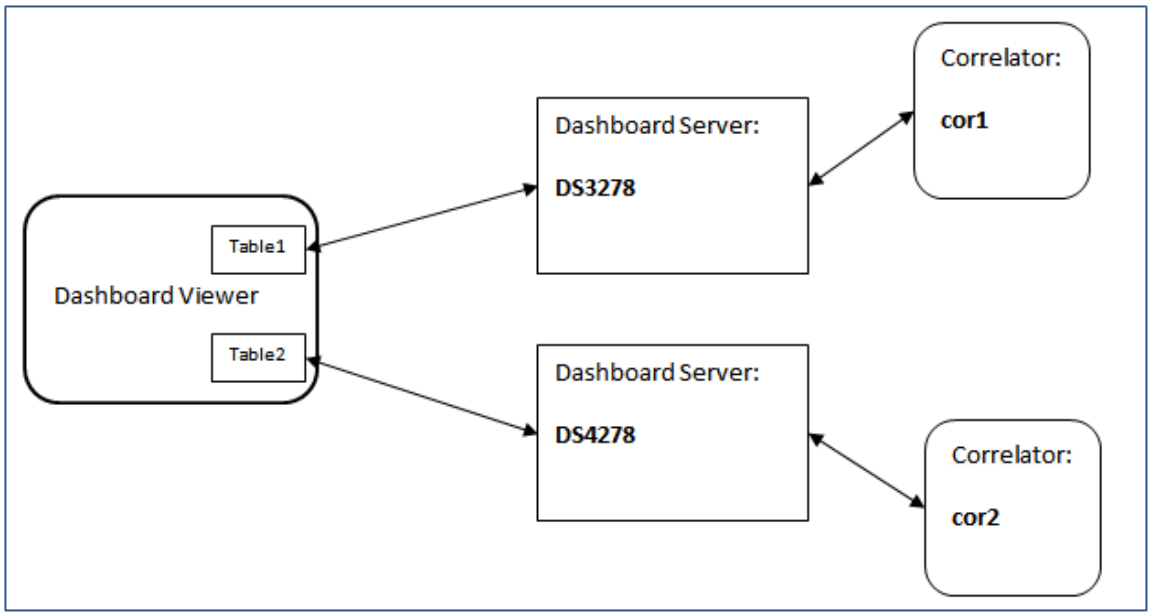

You can override the logical server names specified in the **Application Options** dialog with the `--namedServer` *name* :*host* :*port* option to the Builder executable. Below is an example. This is a sequence of command line options which should appear on a single line as part of the command to start the executable:

```
--namedServer Server1:ProductionHost_A:3278 --
namedServer Server2:ProductionHost_B:4278 --namedServer
Server3:ProductionHost_C:5278
```

Here `Server1`, `Server2` and `Server3` are the server logical names.

### *Viewer with multiple Data Servers*

Viewer maintains connections with the Data Servers named in attachments and commands of opened dashboards.




In the Data Server Login dialog (which appears upon Viewer startup), end users enter the host and port of the default Data Server (or accept the default field values). If all attachments and commands use named Data Servers, end users can check the **Only using named data server connections** check box and omit specification of a default server.

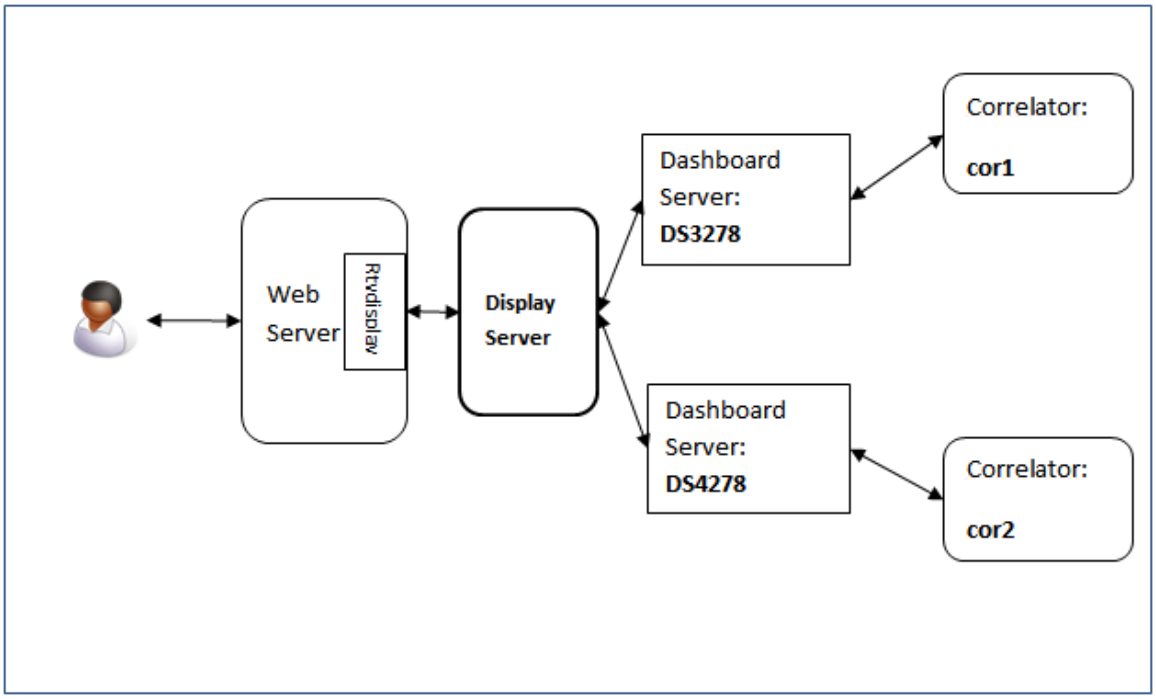
The logical data server names specified in the Builder's **Application Options** dialog are recorded in the deployment package. You can override these logical name definitions with the `--namedServer` *name* `:`*host* `:`*port* option to the Viewer executable. Below is an example. This is a sequence of command line options which should appear on a single line as part of the command to start the executable:

```
--namedServer Server1:ProductionHost_A:3278 --
namedServer Server2:ProductionHost_B:4278 --namedServer
Server3:ProductionHost_C:5278
```

Here `Server1`, `Server2` and `Server3` are the server logical names.

### *Display Server deployments with multiple Data Servers*

The Display Server maintains connections with the Data Servers named in attachments and commands of its client dashboards.




> In a Display Server deployment, each named Data Server must be started with the `--namedServerMode` option.

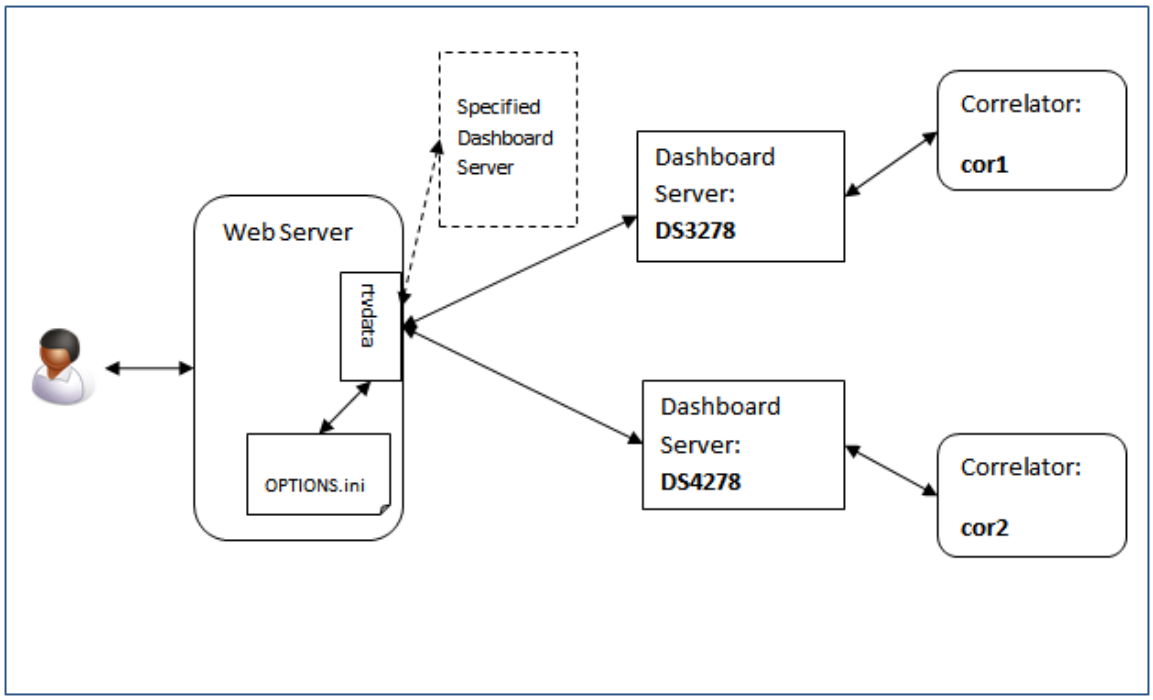
The logical data server names specified in the Builder's **Application Options** dialog are recorded in the file `OPTIONS.ini`, which is used by the Deployment Wizard to define deployment logical names. You can override these logical name definitions with the `--namedServer` *name* `:`*host* `:`*port* option to the Display Server executable. Below is an example. This is a sequence of command line options which should appear on a single line as part of the command to start the executable:

```
--namedServer Server1:ProductionHost_A:3278 --
namedServer Server2:ProductionHost_B:4278 --namedServer
Server3:ProductionHost_C:5278
```

Here `Server1`, `Server2` and `Server3` are the server logical names.

#### *Applet and WebStart deployments with multiple Data Servers*

Applet and WebStart dashboards maintain connections with the Data Servers named in their attachments and commands.




In this diagram, the dotted line indicates the connection to the default Data Server, which is specified in the Startup and Server section of the Deployment Configuration Editor. The default must be running only if some attachments or commands don't specify a named Data Server.

The logical data server names specified in the Builder's **Application Options** dialog are recorded in the file `OPTIONS.ini`, which is used by the Deployment Wizard to define deployment logical names. You can override these logical name definitions with the `--namedServer` *name* `:`*host* `:`*port* option to the Data Server executables. Below is an example. This is a sequence of command line options which should appear on a single line as part of the command to start the executable:

```
--namedServer Server1:ProductionHost_A:3278 --
namedServer Server2:ProductionHost_B:4278 --namedServer
Server3:ProductionHost_C:5278
```

Here `Server1`, `Server2` and `Server3` are the server logical names.

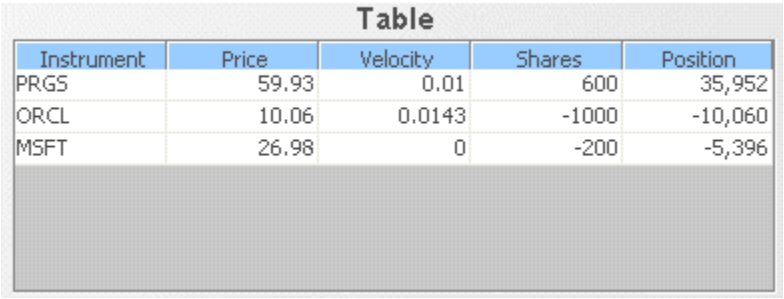
## Using table objects

Table visualizations provide a way to present the contents of data tables in a direct manner. You can present summary information be attaching a table's `valueTable` property to an entire data table, or you can present a specified subset of data table rows and columns by using the filter fields of the **Attach to Apama** dialog.

- Attach the `valueTable` property to a DataView or scenario instance table in order to create an instance summary table.

- Attach the property to a correlator status table in order to display information about each of the correlators that a scenario or DataView connects to.
- Attach the property to a trend or OHLC tables in order to create a tabular display of all the changes to a variable or OHLC values over time.

Double-click **Summary Table** on the tutorial main page to see a typical table object:




In this sample several variables are shown for three instances of a trading scenario. If an end user were to create a new instance of the scenario, it would automatically be added to the table. Each row in the table corresponds to an instance of the scenario.

Table objects support typical table operations such as sorting and column ordering:

- Double-click the header of a column to sort by the column's values. In the table shown above, users can double-click the **Price** column to sort the entries by price.
- Click a column header and drag it to reorder columns.

Sorting large tables can impact dashboard performance, particularly for Display Server deployments. Clear the property `showSortIconFlag` to disable sorting.

Detailed reference information on tables is provided in "Table Objects" on page 411.

## Creating a scenario summary table

Table objects are often attached to a scenario instance table in order to provide a summary view of the instances.

To create a summary table for a scenario, you add a table object to a dashboard and attach its `valueTable` property to a scenario instance table. When you define the attachment, you can select the scenario variables to be displayed; these will be the columns of the table. You can also specify a filter to show only a subset of scenario instances.

Note that, by default, users are authorized to view only those dashboards that they created. Regardless of filter settings, users will not be able to see instances they did not create.

**To create a typical scenario summary table, create a new dashboard and perform the following steps**

1. From the **Tables** tab in the **Object Palette**, select the **Table** object and add it to the dashboard canvas.

2. In the **Object Properties** panel, double-click the `valueTable` property.

   The **Attach to Apama** dialog appears. Attach the table object's `valueTable` property to a scenario instance table.

3. Select the `autoResizeFlag` property and enable it by clicking the check box in the **Property Value** column.

4. Resize the table such that all columns are visible. (You resize the table by selecting it and dragging the handles.)

   The table now displays all the input and output variables of all instances of the specified scenario, as well as the special fields Dashboard Builder adds, including `apama.timestamp` which indicates the time the instance last changed.

   Often, you will not want to display all scenario variables or the special fields in a summary table. The steps that follow show how to specify the variables to be displayed.

5. Double-click the `valueTable` property to display the Attach to Apama dialog.

   By default the display variables field is set to the wildcard "*" indicating that all the variables are to be displayed. Next to the field is a button labeled "…" that provides access to the **Select Columns** dialog.

6. Click on the "…" button to display the **Select Columns** dialog.

7. In the Select Columns dialog select and order the columns.

8. Click **OK** in the **Select Columns** dialog and **OK** in the **Attach to Apama** dialog.

   The table object will now display only those columns you selected.

   By default a table will display a maximum of 100 rows. If a dashboard needs to show more than 100 instances of a scenario, change the value of the `maxNumberOfRows` property. The maximum value for this property is 131072.

   By default a table is unsorted. If you want a table to have a default sort order, set the `sortColumnName` property to the name of the scenario variable to sort by, such as `Price`.

## Filtering rows of a scenario summary table

You can limit the set of instances displayed in a scenario summary by specifying a filter when you define the attachment. This is useful when you only want to display those instances with a shared characteristic, such as the exchange they are trading on.

**To modify a data attachment with filter information**

1. Select the table that you want to modify. For example, double-click **Summary Table** on the tutorial main page, and then select the table object.

2. In the **Object Properties** panel, double-click the `valueTable` property.

3. In the **Attach to Apama** dialog, do the following:

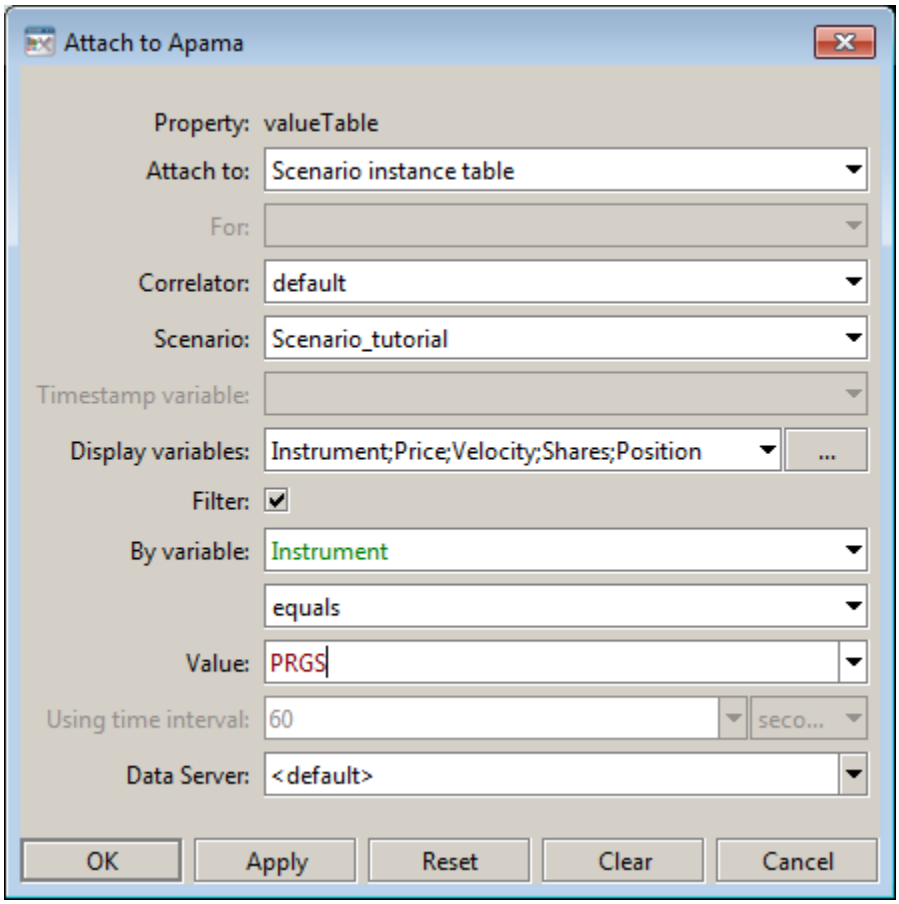
a. Check the **Filter** checkbox.

b. Specify a scenario variable in the **By variable** field.

c. Specify a value or values in the **Value** field. Specify multiple values as a semi-colon-separated list. Do not use spaces.

d. If you specify multiple values, select **Member of** in the field above the **Value** field. (This field specifies a comparison relation. It default to **Equals**.)

   This selects instances whose value for the specified variable bears the specified comparison relation to the specified value. Here is an example:

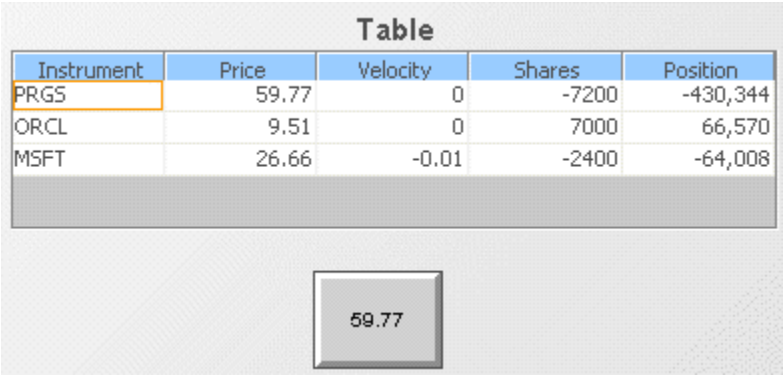



   This example filters the table's contents to display only the instance for which the value of the scenario variable `Instrument` equals `APMA`.

## Performing drilldowns on tables

Frequently you will want to display scenario or DataView summary information in a table and provide the ability to drill down on a single instance or item in order to display detailed information about it. Table objects support drilldowns on a selected row and the passing of substitutions containing the values of one or more variables or fields of the selected instance.

Double-click **Table Drilldown** in the tutorial main page to see the following summary table:

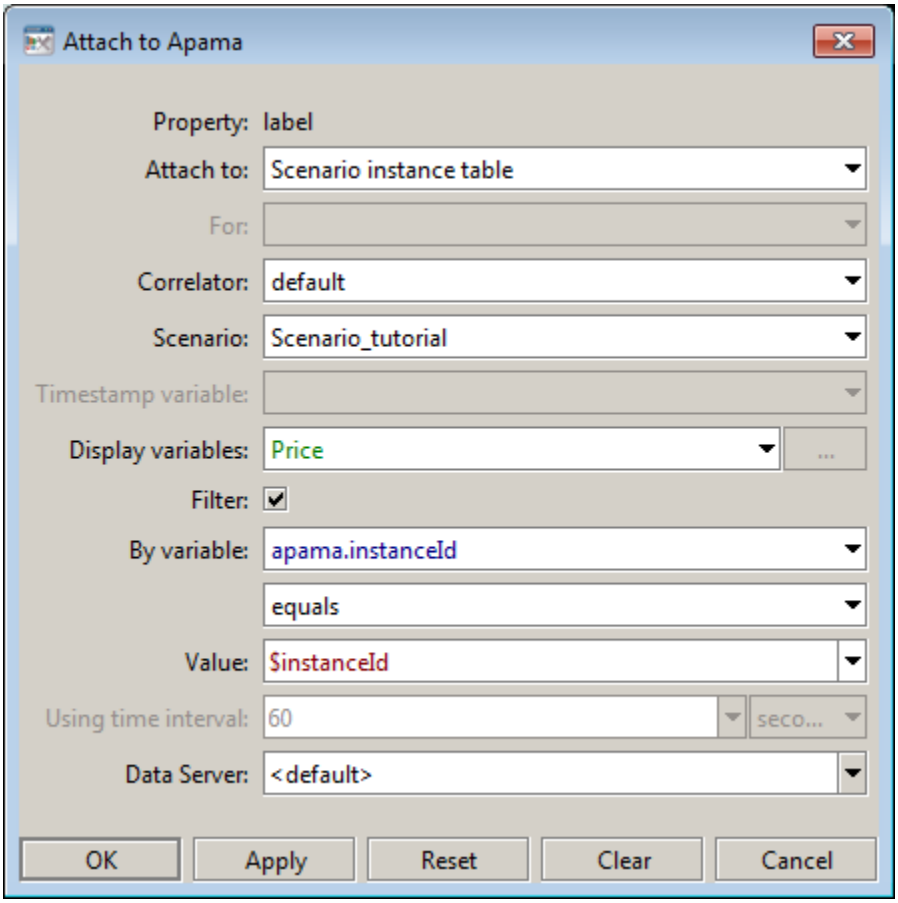



A drilldown has been specified for this table in such a way that the label object updates to show the value of the **Price** variable of the selected scenario instance. As `Price` changes, both the table and label update.

**To specify a drilldown as in the example above**

1. Add a table to a dashboard and attach its `valueTable` property to an instance table as in the previous sample.
2. From the **Labels** tab in the **Object Palette**, select the second label object and add it to the dashboard canvas.
3. Select the label object on the dashboard and in the Object Properties panel double-click the `valueString` property to display the **Attach to Apama** dialog.

   Define the attachment by specifying the **Display** variables and **Filter** fields, for example as follows.




4. Click **OK** in the **Attach to Apama** dialog.
5. Double-click a row in the table. The label object will update to show the value of `Price` for the selected instance.

   The drilldown properties on the table, bar chart, and pie chart objects are preset for the most common usage paradigm where a drilldown on one will redisplay the current dashboard but with new substitution values. This paradigm fits the case where both the scenario summary and instance detail data are displayed in a single dashboard window. You can modify the `drillDownTarget` property on these objects to use a non-default drill-down paradigm, such as displaying detailed information about the selected instance in a separate window. For more information, see "Drill-Down Specification" on page 535.

   In the example above, the label object's data attachment selects the row in the instance table where `apama.instanceId` equals `$instanceId`. This is the most common filter used when performing drilldowns. The drilldown on the table object is defined by default to set the dashboard substitution variable `$instanceId` to the value of `apama.instanceId` for the selected scenario instance. This allows the dashboard that is displayed in response to the drilldown to know which scenario instance it should display data for.

"Specifying drill-down column substitutions" on page 82 describes how to override this default setting.

## Specifying drill-down column substitutions

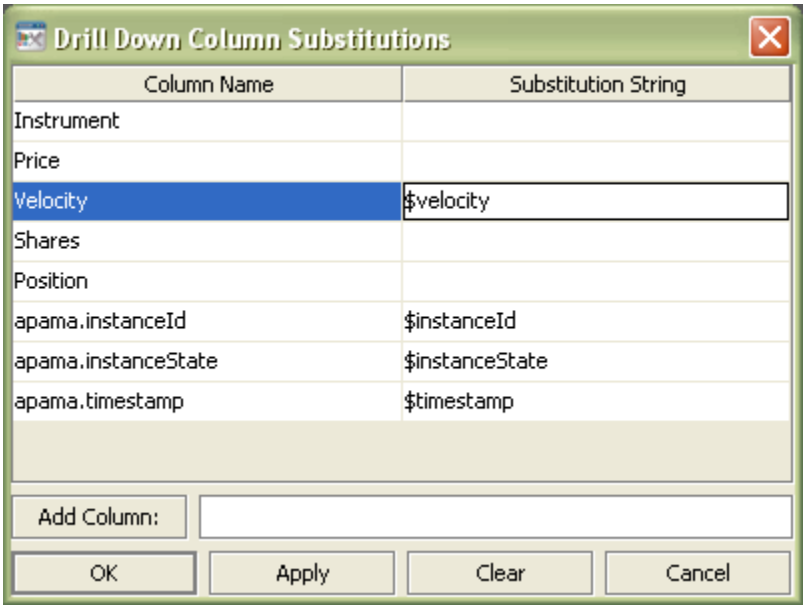
The substitutions set when performing a drilldown on a table object are defined by the `drillDownColumnSubs` property. Here is an example that sets a table column to a dashboard substitution variable, and then attaches a label to the variable.

1. Select the table object and double-click the `drillDownColumnSubs` property.

   The **Drill Down Column Substitutions** dialog appears.

   This dialog allows you to set a substitution variable to the value of a column in the table. By default, table objects are defined to set several substitutions, including `$instanceId` and `$timestamp`. These are set to the values `apama.instanceId` and `apama.timestamp`. In addition, substitutions are inherited by drilldown targets. That is, if a parent object sets a substitution variable for a child (the drilldown target of the parent), then that variable is set the same way for any grandchildren (drilldown targets of the child). You can override these or add additional substitutions with the **DrillDown Column Substitutions** dialog.

2. For the **Velocity** column, set the substitution string field as follows.

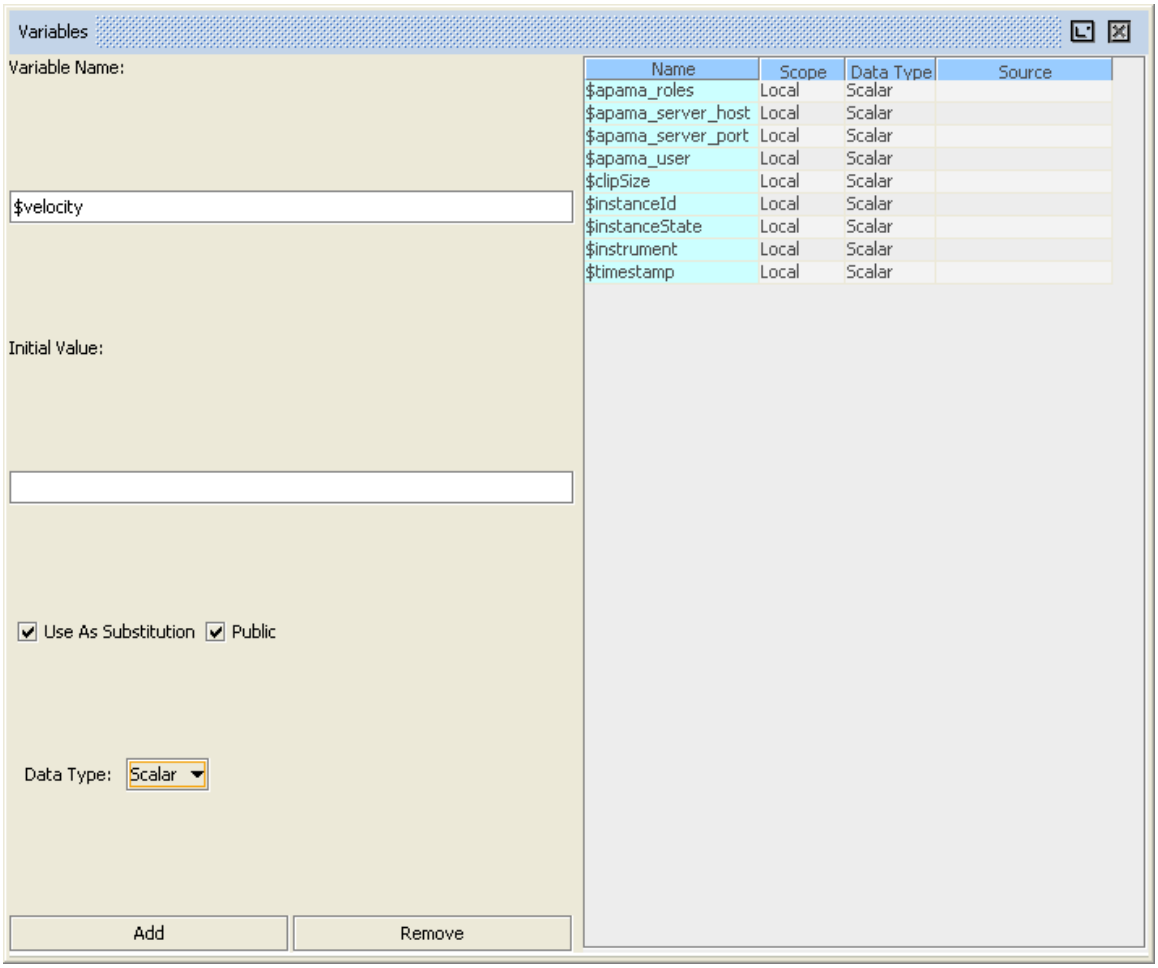



3. Click the **OK** button to close the dialog.

   The substitution variable `$velocity` will now be set when performing a drilldown on the table.

4. Select **Variables** from the **Tools** menu to display the Variables dialog.

   Add the substitution variable `$velocity`.




You must add the `$velocity` substitution variable to the list of local variables, because the drilldown on the table is defined to redisplay the current dashboard. Defining the variable makes it available within the dashboard. If the drilldown displays a different dashboard, the variable must be in that dashboard's list of local variables.

5. Select the label object previously added to the dashboard.
6. In the **Object Properties** panel, right-click the `valueString` property and select **Attach to Data | VARIABLE**.

   This will display the **Attach to Local Variable Data** dialog.
7. Select `$velocity` in the dialog and click **OK**.

   The label is now attached to `$velocity`. When you double-click a row in the table, the dashboard performs the drilldown and sets `$velocity` to the current value of `Velocity` of the selected scenario instance. The dashboard updates the label object to show this value.

Note that when a visualization object is attached directly to Apama, it updates whenever the corresponding scenario variable or DataView field changes; but when it is attached to a dashboard substitution variable, it does not.

If you want a dashboard's visualization objects to update as scenario variables or DataView fields change, attach them directly to Apama using `$instanceId` in the filter.

If you do not want the objects to update, that is, if you want only the values *at the time drilldown was performed*, define a drill down substitution to set a substitution variable to the current value, and then use that substitution in the dashboard drilled down to.

## Hiding table columns

When you define drilldown substitutions on a table object, only those variables selected as the display variables in the table's data attachment are available for setting substitution values. In the previous example, if the `Velocity` variable was not selected as a display variable for the table, then it would not have been available as a column in the **Drill Down Column Substitutions** dialog.

If you have a scenario variable or DataView field that you want to use to set a substitution when performing a drilldown on a table but do not want to appear as a column in the table, include it as a display variable when defining the attachment and set the `columnsToHide` property to prevent it from being displayed. To hide multiple variables specify them as a semicolon-separated list.

The `columnsToHide` property is preset to hide the `apama.instanceId` column. Apama transparently forces `apama.instanceId` to be included as a display variable on all table objects. This is so that you perform a drilldown, `$instanceId` can be set to the ID of the selected instance. You should always hide the `apama.instanceId` column.

## Using pre-set substitution variables for drill down

There are some hidden variables that are always set when you perform a drilldown. These are useful if you want to know which column or cell was selected to perform the drilldown:

- `$celldata`: Set to the value of the cell selected.
- `$colName`: Set to the name of the column of the cell.

You can use these variables, for example, as parameters to functions or commands whose action you want to vary based on the column or cell value selected.

## Formatting table data

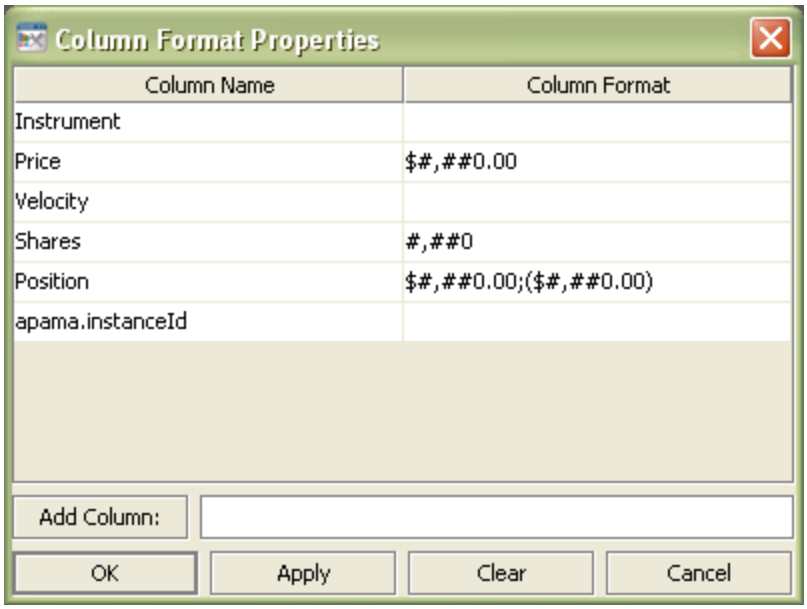
The table object allows formatting attributes to be specified for each column in a table. Double-click **Formatted Table** in the tutorial main page to see the following table:

Table

| Instrument | Price | Velocity | Shares | Position |
|---|---|---|---|---|
| PRGS | $59.77 | 0 | -2,000 | ($119,540.00) |
| ORCL | $9.43 | -0.0167 | 19,600 | $185,024.00 |
| MSFT | $27.05 | 0 | -17,800 | ($481,668.00) |

Here formatting has been specified for the `Shares`, `Price`, and `Position` columns. The `Price` and `Position` columns include a currency indicator and the `Position` column is presenting negative positions inside parenthesis. Apama dashboards provide wide variety of formats, and you can specify custom formats as well.

To specify formatting information, double-click the `columnFormat` property and use the **Column Format Properties** dialog:




To format a column, select the column in the **Column Name** field and either select, or type, a format string in the **Column Format** field.

Specify column formats using a format string appropriate for use with the Java class `java.text.DecimalFormat`, or with the following shorthand: $ for US dollar money values, $$ for US dollar money values with additional formatting, () for non-money values, formatted similar to money, or # for positive or negative whole values.

## Colorizing table rows and cells

The table object allows the color attributes of rows and cells to be set based on the value of a scenario variable or DataView field. Double-click **Colored Table** on the tutorial main page to see the following table, which shows a typical use for setting color attributes.

Table

| Instrument | Price | Velocity | Shares | Position |
|---|---|---|---|---|
| PRGS | $59.26 | 0 | 3,200 | $189,600.00 |
| ORCL | $9.07 | -0.0143 | 30,600 | $277,542.00 |
| MSFT | $26.87 | 0 | -28,400 | ($763,108.00) |

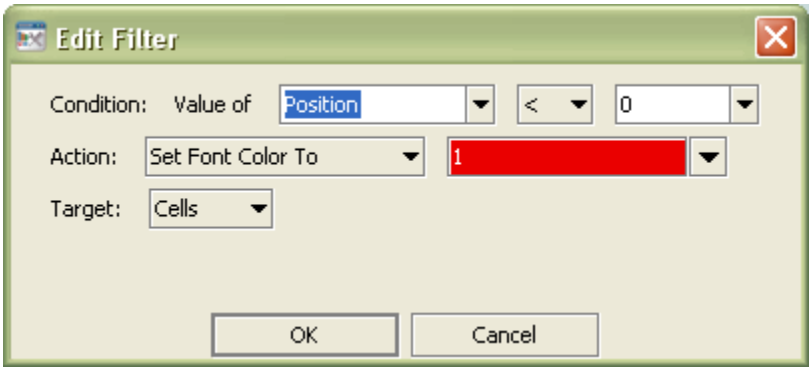
Here the Position cell is shown with green text if the position is positive and red text if it is negative. Colorizing a table can make it much easier to identify values of interest. Colorizing attributes are specified by setting the `filterProperties` property.

**To set the `filterProperties` property**

1. Double-click the `filterProperties` property.

   The Filter Properties dialog appears.

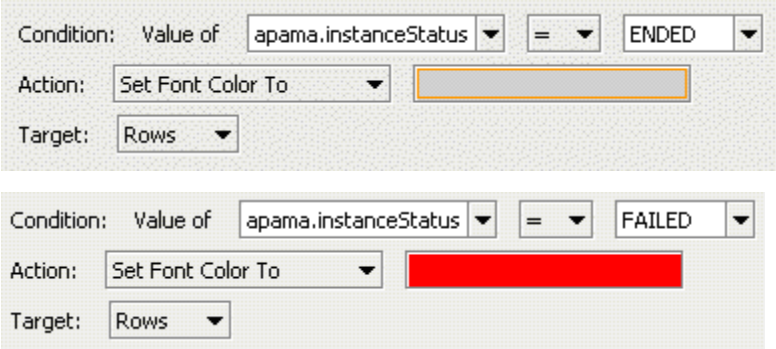
2. Double-click a filter to edit it or click the **Add** button to add a new filter.




   Here the filter specifies that the font color of the `Position` cell should be red if the value of Position is less than 0. The **Condition** fields allow you to specify the condition which must be matched for the action to take affect. The **Action** field allows you to set the font or background color or hide a row. Hiding a row is useful if you do not want the row to appear based on some attribute of the scenario instance. The **Target** field allows you to apply the action to single cell, row, or column.

A common use of table colorization is to provide a visual indication of the scenario instances which have ended or failed. For example you may want to set the font color to gray for those which have ended and red for those which have failed.

To do this you must include `apama.instanceStatus` as a display variable in the table's data attachment and, typically, in the list of `columnsToHide`. The filter properties for the table can then be used to set the font color based on the value of `apama.instanceStatus` with the following two filters. The following illustrations show how the Edit Filter dialog can be used for this purpose.




## Setting column headers

By default the header for each column is the name of the scenario variable or DataView field it shows.

**To change column headers by setting the `columnDisplayNames` property**

1. Select a table object in the **Object Properties** panel.
2. Double-click the `columnDisplayNames` property.

   The **Column Display Name Properties** dialog appears.

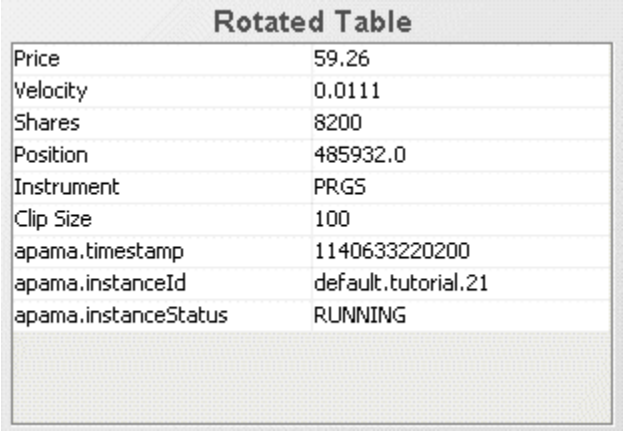
   The header for the `Instrument` column is set to `Stock Symbol`.
3. Enter the desired column names in the dialog. If you want the header to span multiple lines include `\n` in the display name such as "`Stock\nSymbol`".

## Using rotated tables

Rotated tables rotate the data in the data table they are attached to such that rows become columns and columns become rows.

**To create a rotated table**

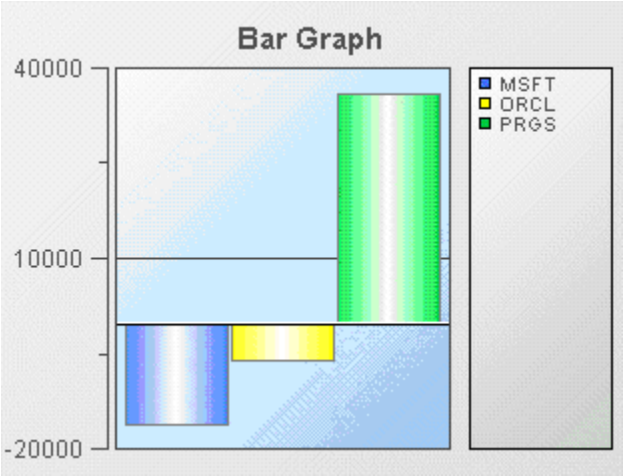
1. From the **Tables** tab in the **Object Palette**, select the **Rotated Table** object and add it to the dashboard canvas.
2. Attach the table object's `valueTable` property to scenario data just as you would for other kinds of tables (see, for example, "Creating a scenario summary table" on page 77).



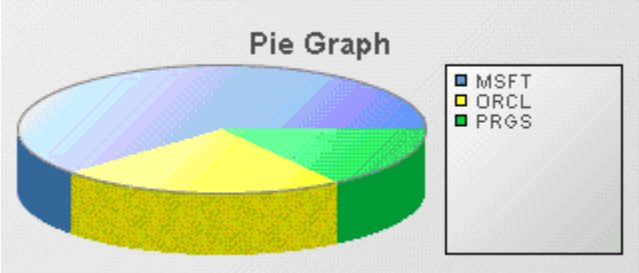

Here the rotated table is attached to a scenario instance table, and the filter is set to select only the instance where **Instrument** equals **APMA**. Without a filter, all instances of the scenario appear as columns.

## Using pie and bar charts

Pie and bar charts can be used in dashboards as an alternative to table objects for showing scenario or DataView summary data. The charts are similar in their configuration and behavior. The following illustration shows a typical bar chart:




The following illustration shows a typical pie graph.



Both the bar and pie charts shown above display the value of `Position` for each scenario instance. The bar chart provides an indication of negative values but the pie chart does not. Each chart supports drill downs similar to those supported by table objects.

Detailed reference information on graphs, including pie and bar charts, is provided in "Graph Objects" on page 265.

## Creating a summary pie or bar chart

To create a summary pie or bar chart for a scenario or DataView, you add an instance of the object to a dashboard and attach its `valueTable` property to a DataView or scenario instance table. When you define the attachment, you can select the variable to be charted as well as the label to be used for the data in the chart legend. As with table objects, when you define the data attachment, you can also supply a filter that specifies the subset of the instances that are charted.

Note that users can view only those scenario and DataView instances that they created. Regardless of filter settings, users will not be able to see instances they did not create.

To see a sample bar chart, double-click **Bar Chart** on the tutorial main page.

**To create a summary bar chart, create a new dashboard and perform the following steps**

1. From the **Graphs** tab in the object palette, select the Bar Graph object and add it to the dashboard canvas.
2. With the graph object selected, double-click the `valueTable` property in the **Object Properties** panel, and attach the graph to a scenario.
3. Click **OK**.

The bar chart will now chart the value of `Position` for each instance of the scenario.

In this example the display variables were set to `Instrument` and `Position`. `Instrument` is a string variable and was included to provide a meaningful label for each bar in the chart legend.

For both bar and pie charts, you can pick a scenario string variable to use as the label in the legend. Do not pick a number variable as it will be interpreted as the value to chart.

If multiple number variables are selected for display, the behavior is controlled by the `rowSeriesFlag` property.

## Using series and non-series bar charts

Data in bar charts can be displayed as both series and non-series data. This is determined by the `rowSeriesFlag` property.

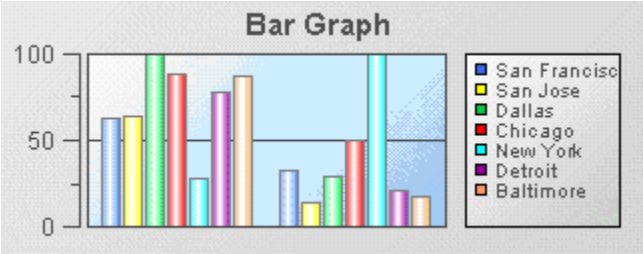
If the `rowSeriesFlag` property is enabled, one group of bars will be shown for each numeric column in the data attachment. Within the group for each numeric column, there will be a bar for each row in that column. Column names will be used for the x-axis labels. If your data attachment has a label column and the `rowLabelVisFlag` is selected, data from this column will be used in the legend. If your data attachment does not have

a label column, select the `rowNameVisFlag` checkbox to use row names in the legend. By default, the label column is the first non-numeric text column in your data. Specify a column name in the `labelColumnName` property to set the label column to a specific column.

If the `rowSeriesFlag` property is not enabled, one group of bars will be shown for each row in your data attachment. Within the group for each row, there will be a bar for each column in that row. Column names will appear in the legend. If your data attachment has a label column and the `rowLabelVisFlag` is selected, data from this column will appear on the x-axis. If your data attachment does not have a label column, select the `rowNameVisFlag` checkbox to use row names on the x-axis. By default, the label column is the first non-numeric text column in your data. Specify a column name in the `labelColumnName` property to set the label column to a specific column.

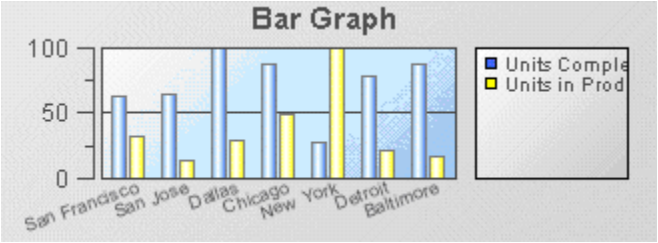
1. Create a new dashboard, select the **Graphs** tab in the Object Palette, and add a Bar Graph object to the dashboard canvas.

   By default the `rowSeriesFlag` property is enabled and the chart appears as follows.




2. With the graph object selected, in the **Object Properties** panel, select the `rowSeriesFlag` property and disable it.
3. Select the `xAxisFlag` property and enable it.

   The chart will now appear as follows.



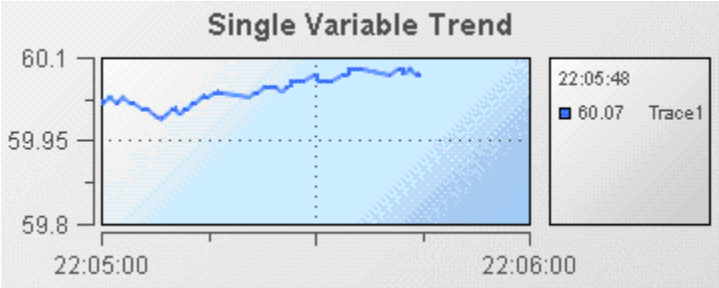

## Performing drilldowns on pie and bar charts

Drilldowns on pie charts are defined by setting the same properties you set on table objects in order to perform a drilldown: `drill down target` and `drillDownColumnSubs`.

## Using trend charts

Trend charts are present in **Trends** and **Trends HTML5** tabs of the object palette. The trend charts in the **Trends HTML5** tab provide a pure HTML implementation of a trend chart. The HTML version of a trend chart provides an interactive and high performance trend chart in an HTML5 compatible browser.

Trend charts provide the ability to view changes in a scenario variable or DataView field over time. The following illustration shows a typical trend chart:



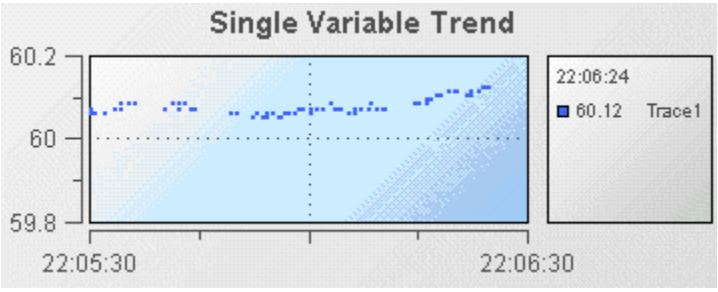

In this sample, a single trend line is displayed to show the value of the `Price` variable of an instance of the tutorial scenario.

A trend chart can display up to ten trace lines allowing you to compare changes in up to ten scenario variables. Useful examples of trend charts might show the changes in price for two stocks or the movement of a single stock price relative to a market average.

The traces in a trend chart can be shown as lines or as individual data points.

1. Open the file `tutorial-trend.rtv` by selecting **Trend Chart** on the tutorial main page.
2. Select the trend chart and in the Object Properties panel select the property `trace1MarkStyle` and change its value to `1`.
3. Select the property `trace1LineStyle` and change its value to `0`.

   The trace line in the trend chart will now be displayed as a series of points.

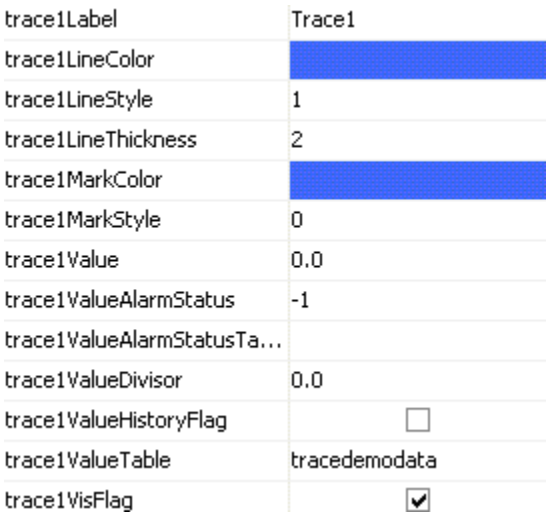



The data values displayed are the same; only the presentation has changed.

Detailed reference information on trend charts is provided in "Trend graphs" on page 497.

## Creating a scenario trend chart

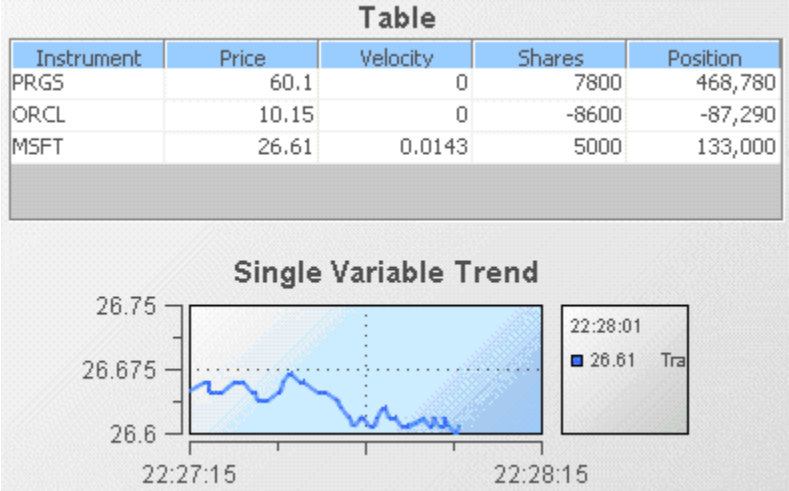
To create a trend chart for a scenario, you add it to a dashboard and set its `traceCount` property to the number of trace lines you want to display. This will cause a set of properties to be added to the property panel for each trace; `trace1` through `trace`*N*. Following are the properties for `trace1`.




Each trace will have a `trace`*N*`Value` and `trace`*N*`ValueTable` property. These define the data attachment for the traces. The `trace`*N*`Value` property is used to attach the trace to new data (data received after the time of attachment). The `trace`*N*`ValueTable` property is used to attach the trace to historical data (data received before the time of attachment).

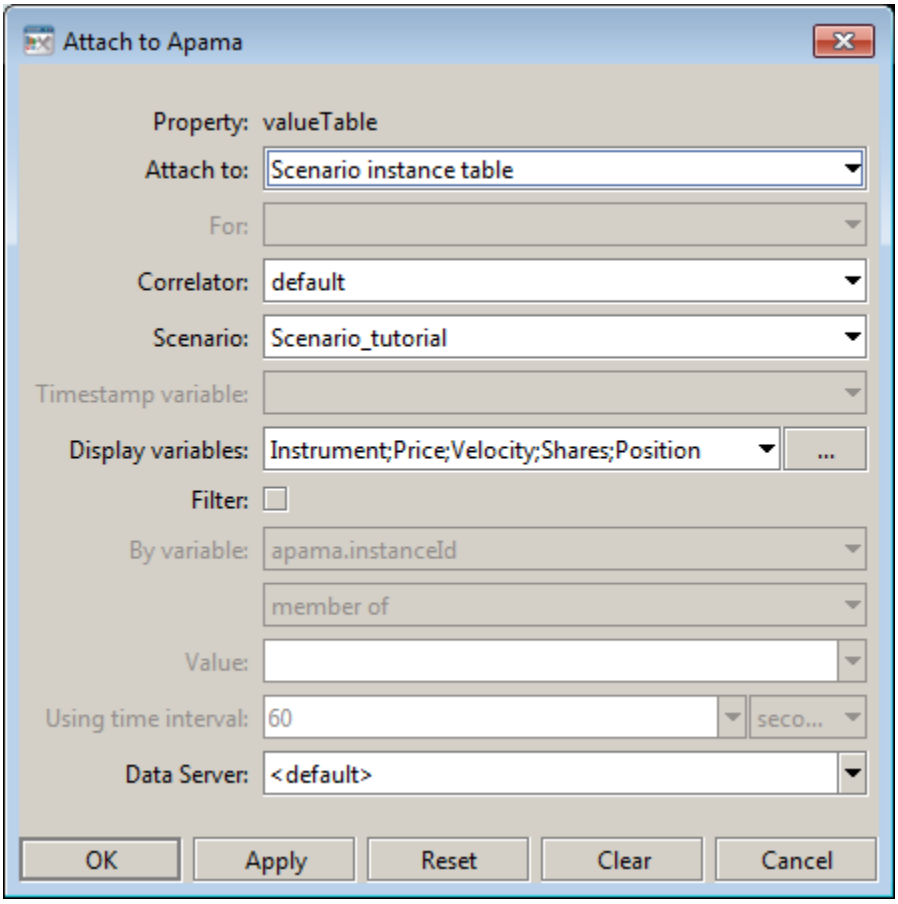
When attaching a trace to a scenario variable, you must specify a filter that identifies the scenario instance the trace will show data for. The filter to identify the instance will typically match on `$instanceId` although other filters can also be used.

The Trend Drilldown tutorial sample demonstrates how to use a trend chart where the scenario instance charted is determined by the selection in a scenario summary table. The following illustration is from the Trend Drilldown sample.




**To recreate this sample, create a new dashboard and perform the following steps**

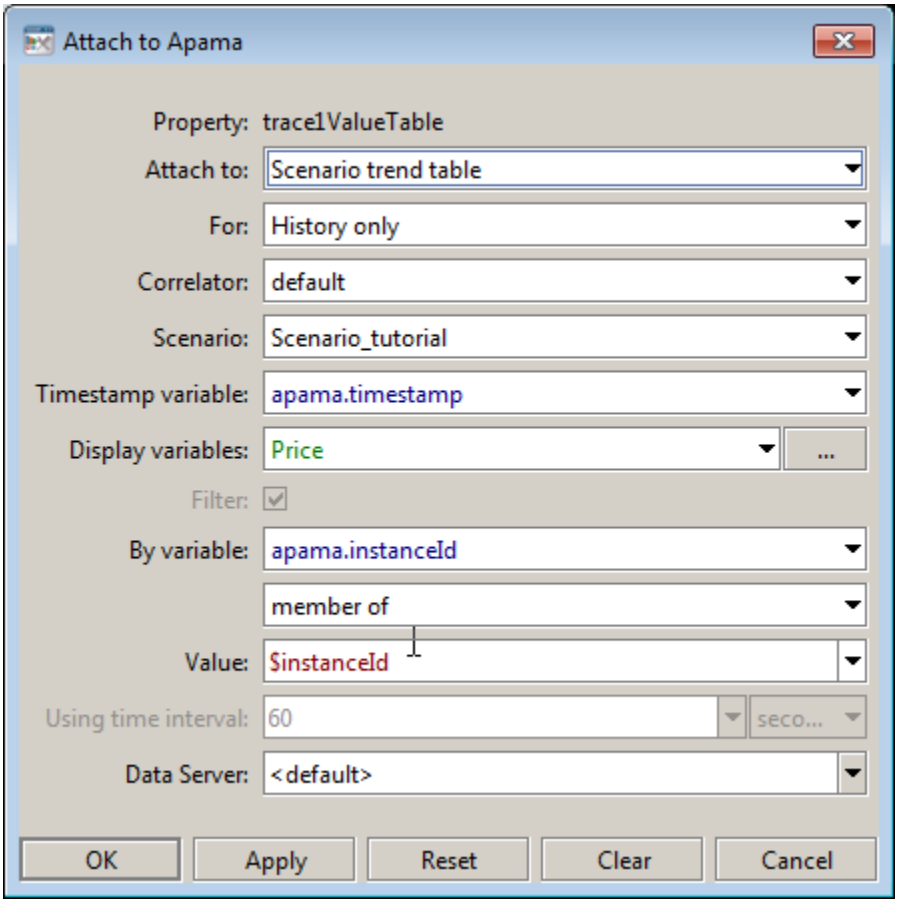
1. From the **Tables** tab in the Object Palette, select the **Table** object and add it to the dashboard canvas.
2. With the table object selected, in the **Object Properties** panel, double-click the `valueTable` property and attach it to Apama by specifying the information shown below in the **Scenario** and **Display variables** fields. Do not apply a filter.




3. From the **Trends** tab in the Object Palette, select the **Single Variable Trend** object and add it to the dashboard canvas.

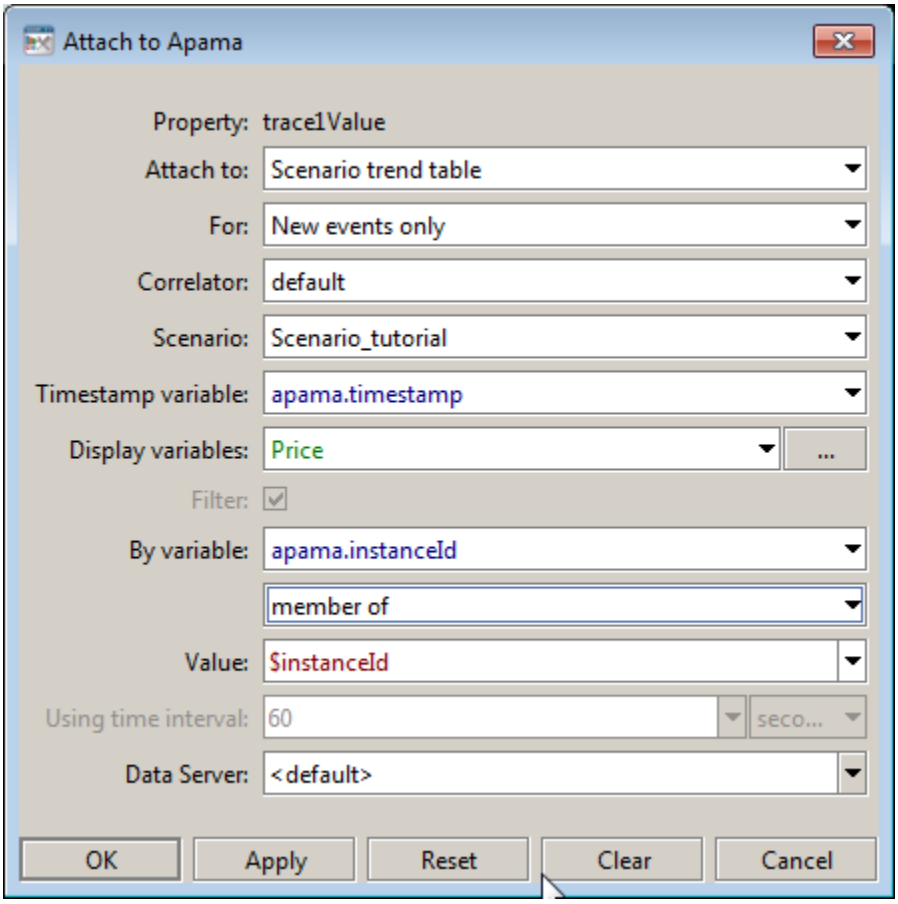
4. With the trend object selected, in the Object Properties panel, double-click the trend chart object's `trace1ValueTable` property and attach it to the trend table for the tutorial scenario by specifying values in the fields as shown below.

   Here the `Price` variable is selected for the trace. The scenario instance charted will be the selected instance as indicated by the variable `$instanceId`.




5. With the trend object selected, in the Object Properties panel, double-click the trend chart object's `trace1Value` property and attach it to the trend table for the tutorial scenario by specifying values in the fields as shown below.

   Here the `Price` variable is selected for the trace. The scenario instance charted will be the selected instance as indicated by the variable `$instanceId`.



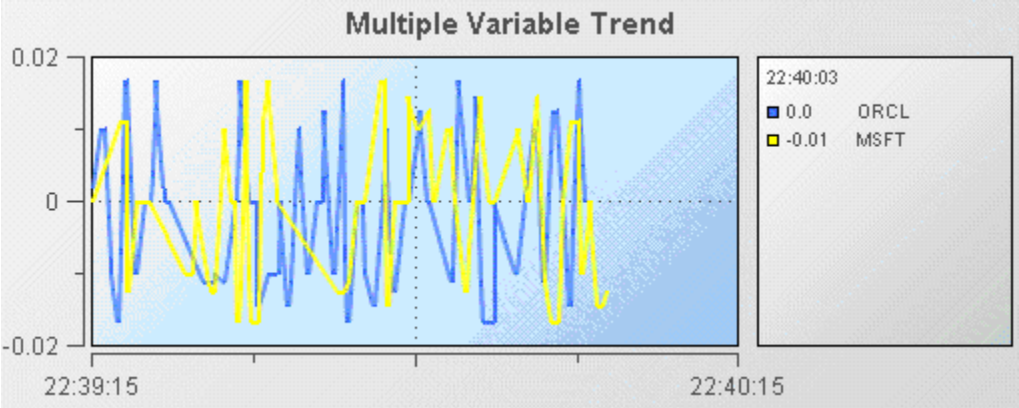

6. Select the trend object's `scrollbarMode` property and change its value to `As Needed`. This will add a scrollbar to the chart allowing you to scroll back in time to view earlier values.

7. Select a scenario instance in the table by double-clicking on it. The chart will now begin charting the `Price` variable of the selected scenario instance.

   If you have not previously displayed a sample containing a trend chart, no previous values for `Price` will be displayed. Apama does not collect data in a scenario trend table until the first attachment to an instance of the table is made.

## Charting multiple variables

Trend charts are able to show up to 10 trace lines. This is useful for comparing changes in the values of multiple variables or fields. The following illustration shows the multiple variable trend chart from the Multiple Trend Lines tutorial sample.



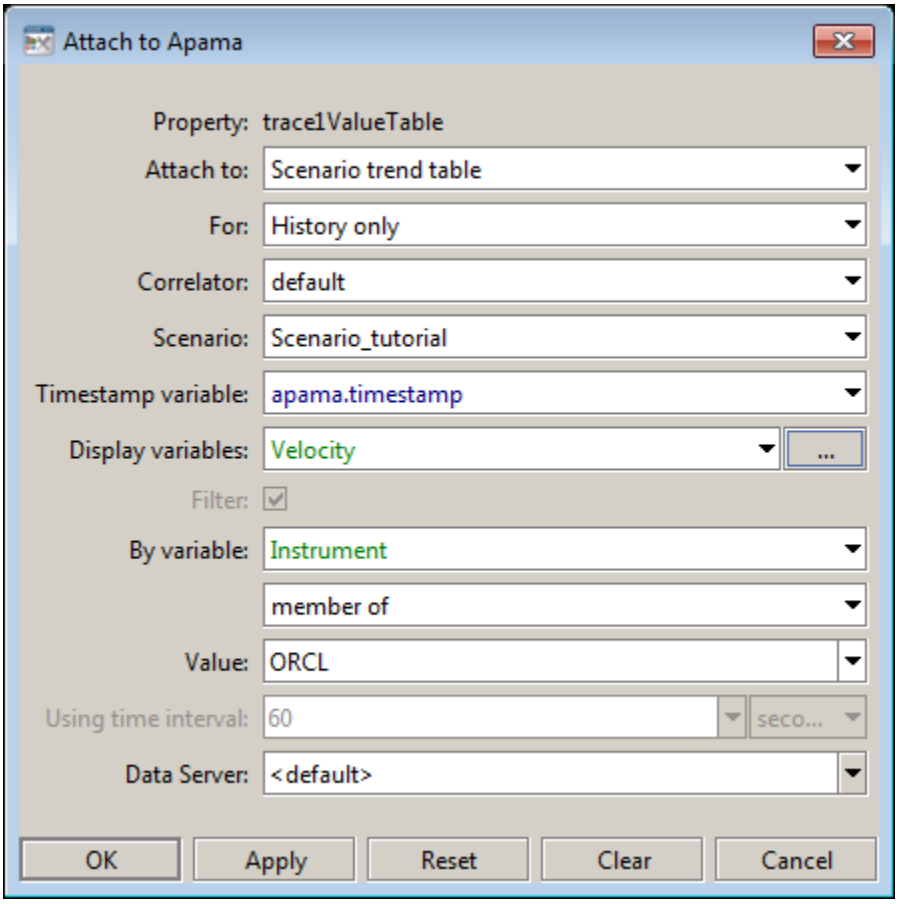

Here the trend chart displays the `Velocity` of the stock price of two instances of the tutorial scenario; one where the Instrument is ORCL and the second where the Instrument is MSFT.

**To recreate this sample, create a new dashboard and perform the following steps**

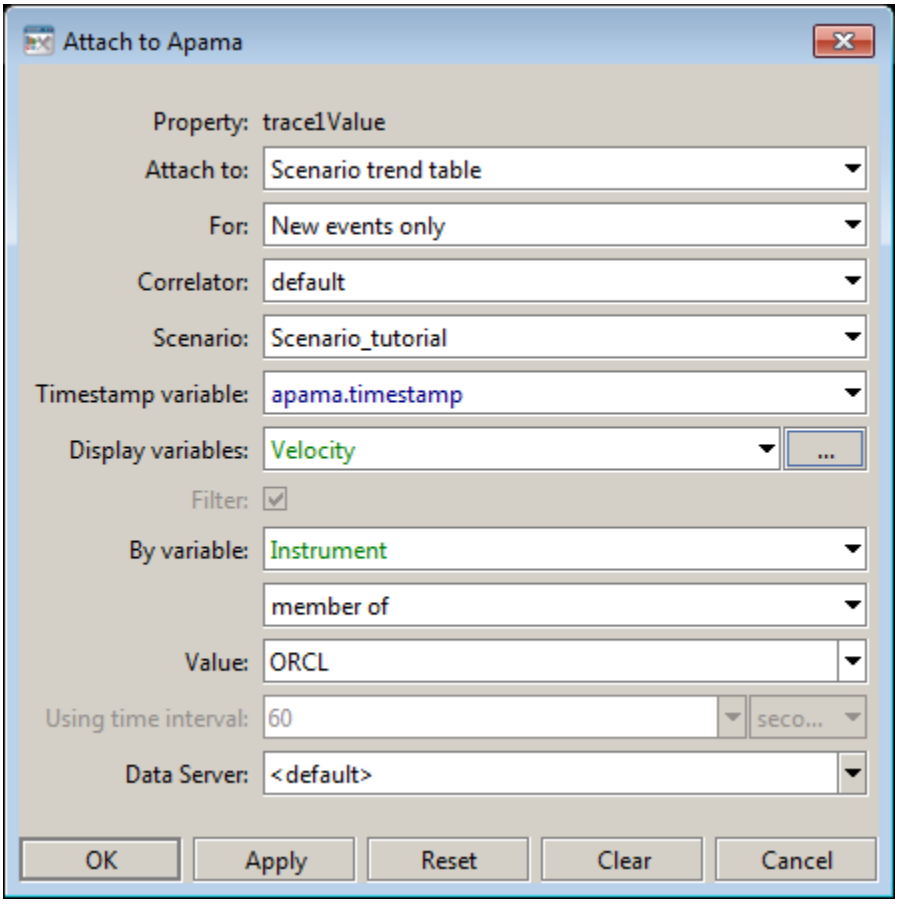
1. From the **Trends** tab in the object palette, select the Multiple Variable Trend object and add it to the dashboard canvas.

   The multiple and single variable trend objects are virtually the same. The only difference is that in the multiple variable trend object the `traceCount` property is set to 2. If you need to display more than two trace lines you can select either object and set the `traceCount` property to the number of traces needed.

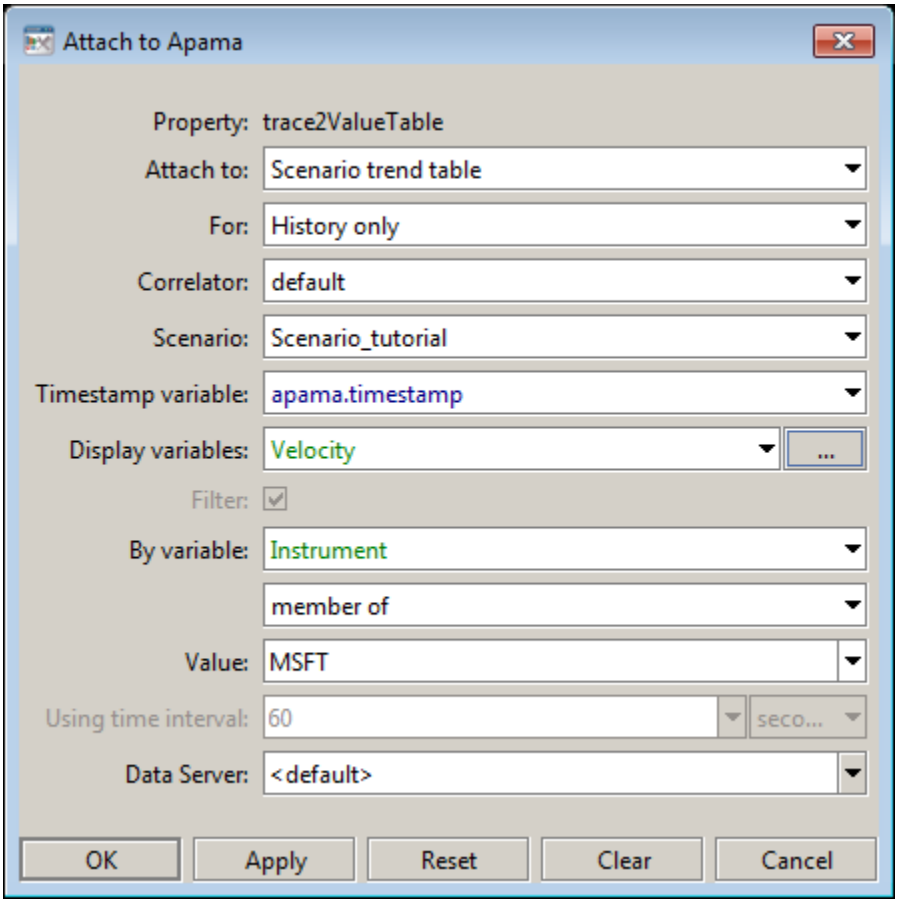
2. With the trend object selected, in the Object Properties panel, double-click the trend object's `trace1ValueTable` property and attach it to Apama by specifying the following information:




3. With the trend object selected, in the Object Properties panel, double-click the trend object's `trace1Value` property and attach it to Apama by specifying the following information:

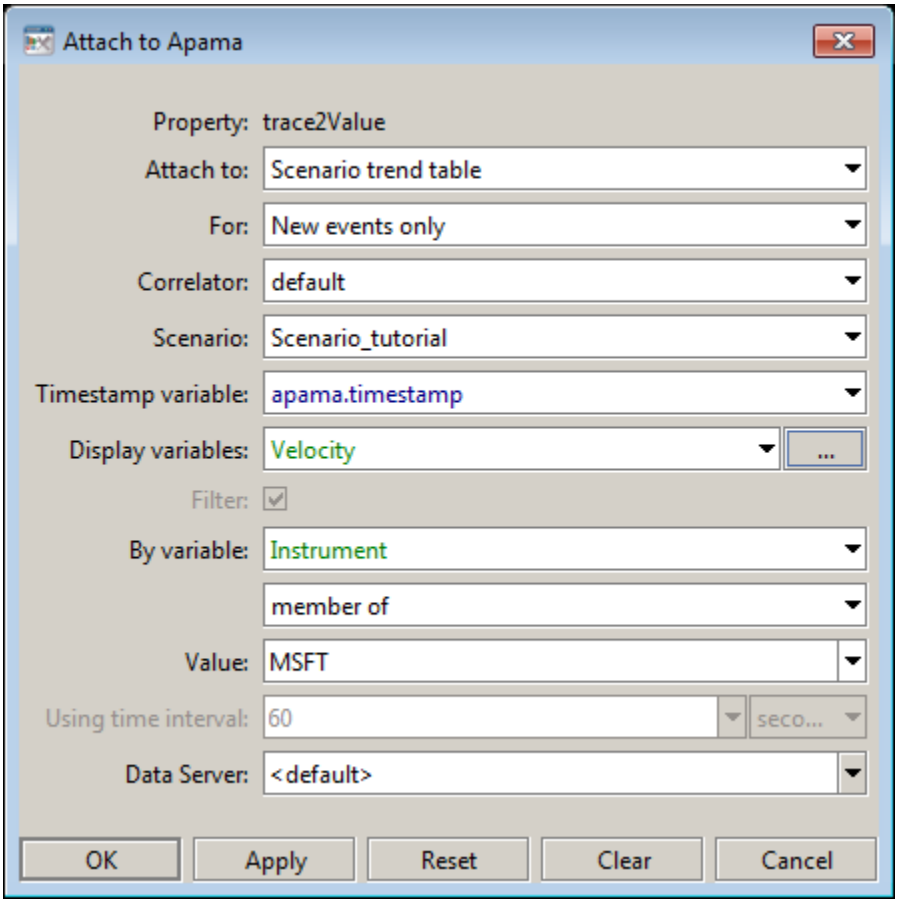



4. Attach the `trace2ValueTable` property to Apama by specifying the following information:



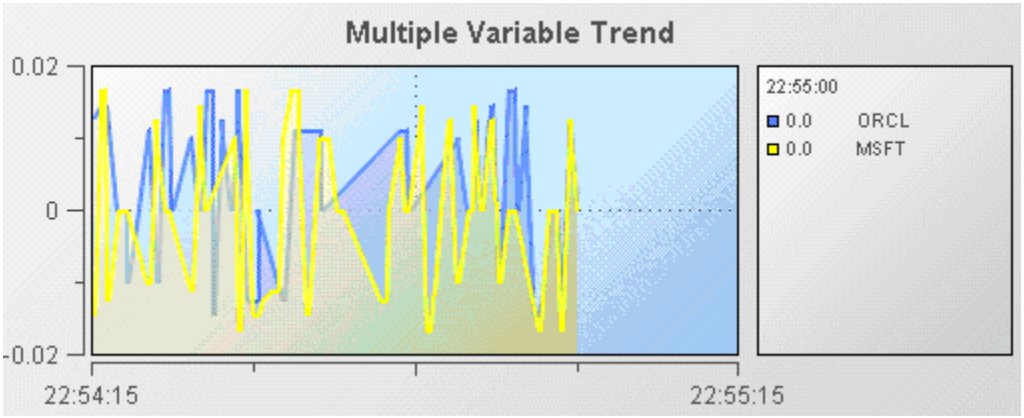

The trend chart will now chart the `Velocity` variable of the instances of the tutorial scenario which match the filters where `Instrument` equals `ORCL` and `MSFT`.

5. Attach the `trace2Value` property to Apama by specifying the following information:



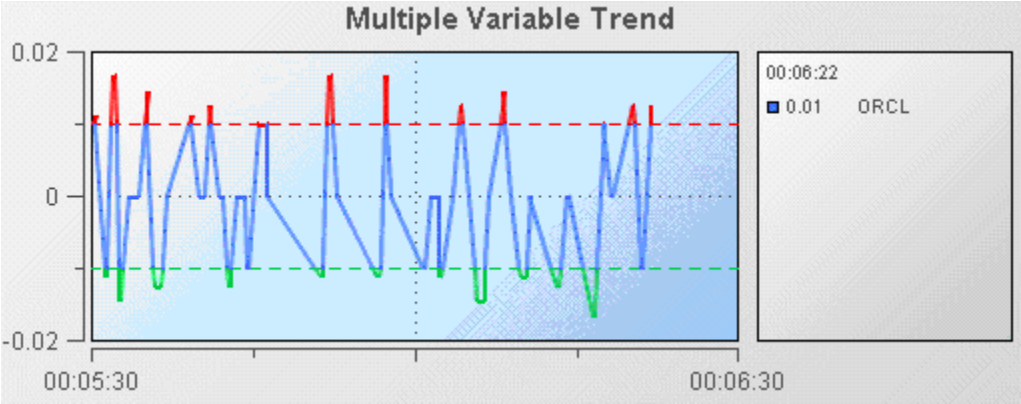

The trend chart will now chart the `Velocity` variable of the instances of the tutorial scenario which match the filters where `Instrument` equals `ORCL` and `MSFT`.

When displaying multiple traces, it is often useful to display them as filled regions. To specify this, select the `traceFillStyle` property and change its value to `Transparent Gradient`. The following illustration shows an example of a trend chart with filled regions.



## Adding thresholds

Often you will want to know when the value of a scenario variable or DataView field is outside a specified range. For example you may want to know when the price of a stock is above or below some threshold. Trend charts enable you to display thresholds and show when variables cross them. The following illustration from the Trend Thresholds tutorial sample shows a typical example.



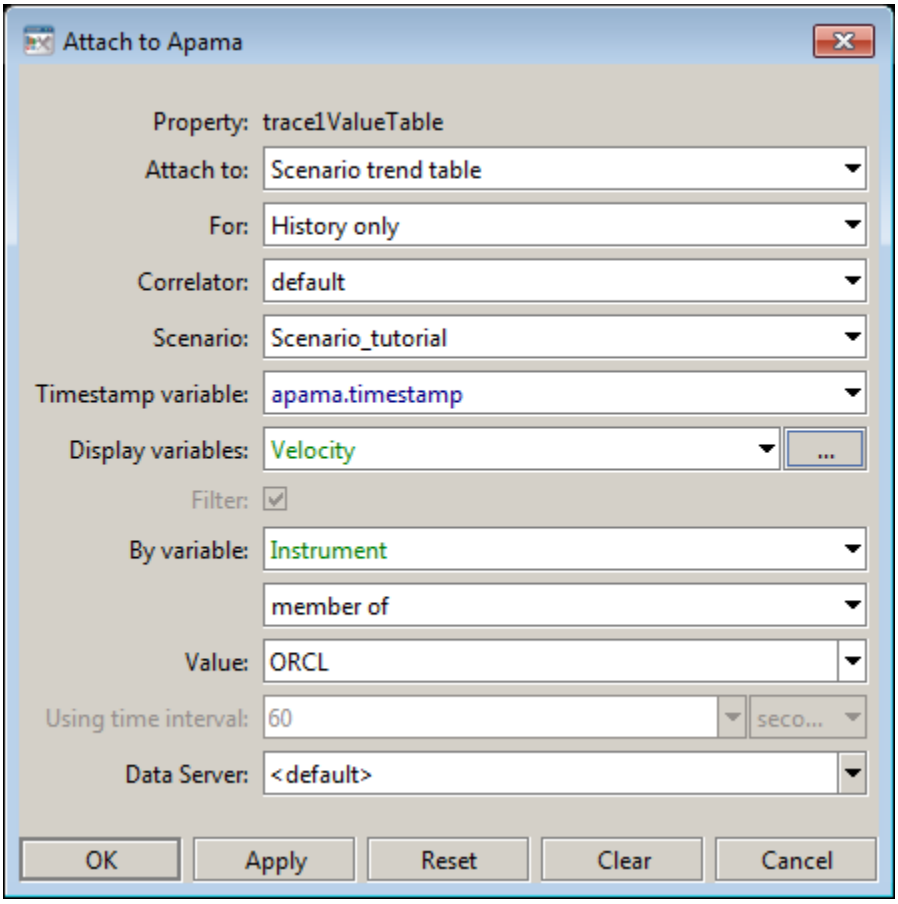

Here the `Velocity` of an instance of the tutorial is being charted and high and low thresholds of .01 and -.01 are being displayed.

Trend charts support four thresholds that are specified with the properties; `valueHighAlarm`, `valueLowAlarm`, `valueHighWarning`, and `valueLowWarning`. These properties can be set to fixed values or attached to scenario variables. Each threshold has a set of properties for configuring it. Following are the properties for the `valueHighAlarm` property.

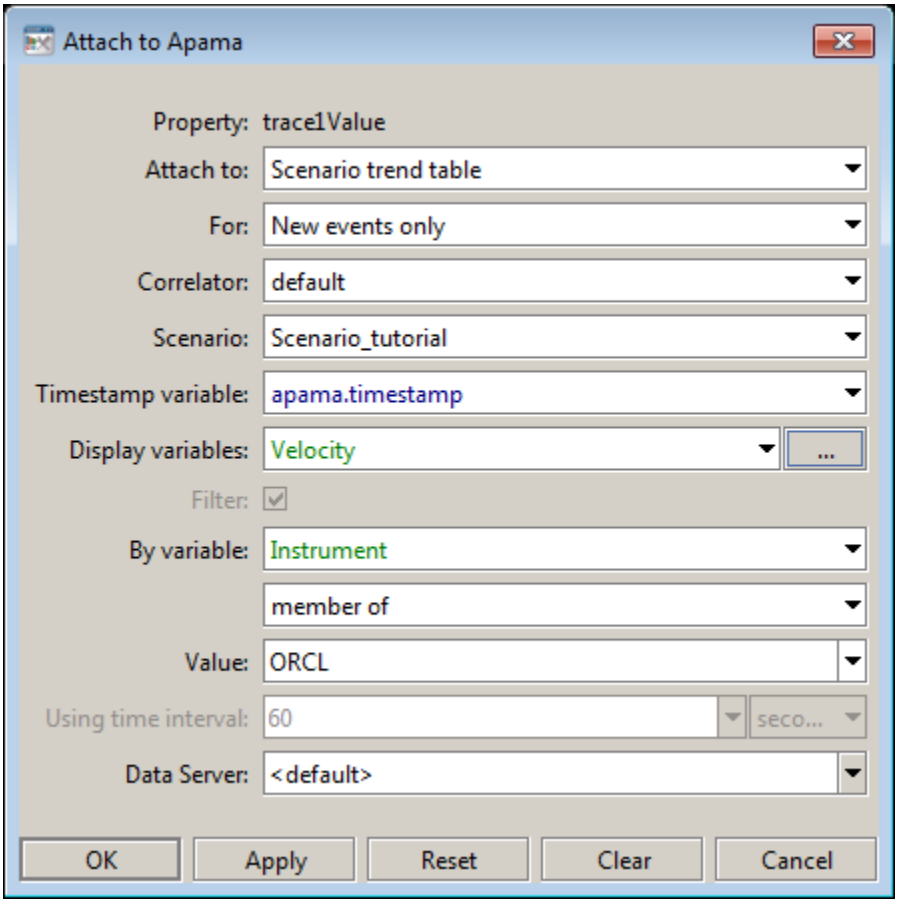
| | |
|---|---|
| valueHighAlarm | 0.01 |
| valueHighAlarmEnabledFlag | ☑ |
| valueHighAlarmLineVisFlag | ☑ |
| valueHighAlarmMarkColor | |
| valueHighAlarmMarkStyle | 0 |
| valueHighAlarmTraceColor | |
| valueHighAlarmTraceStyle | 1 |

**To recreate this sample create a new dashboard and perform the following steps**

1. From the **Table** tab in the Object Palette, select the Threshold Trend object and add it to the dashboard canvas.
2. With the threshold trend object selected, in the Object Properties panel, select the `trace1ValueTable` property and attach it to Apama by specifying the following information:




3. With the threshold trend object selected, in the Object Properties panel, select the `trace1ValueValue` property and attach it to Apama by specifying the following information:




The trace line will now show the `Velocity` of the instance of the tutorial scenario where Instrument equals ORCL.

4. Select the `valueHighAlarm` property and change its value to `0.01`.
5. Select the `valueLowAlarm` property and change its value to `-0.01`.

Thresholds will now be displayed.

## Configuring trend-data caching

By default, dashboard servers (Data Servers and Display Servers) collect trend data for all numeric output variables of scenarios and data views running in their associated correlators. This data is cached in preparation for the possibility that it will be displayed as historical data in a trend chart when a dashboard starts up. Without the cache, trend charts would initially be empty, with new data points displaying as time elapses.

Advanced users can override the default caching behavior on a given server, and control caching in order to reduce memory consumption on that server, or in order to cache variables that are not cached by default, such as non-numeric variables.

For more information, see "Configuring Trend-Data Caching" on page 663.

## Rendering trend charts in HTML5

A dashboard trend graph object can be rendered by using HTML5, which provides added functionalities to the chart object without requiring a Flash Player or other browser plug-in. Select the `webChartFlag` option to enable this feature. Note: this feature only applies to display server dashboard deployment.

The `webChartFlag` option appears in the list of properties for trend graphs (`obj_trendgraph02`).

When `webChartFlag` is checked, the HTML5 rendering of the trend graph object appears in place of the swing trend graph in the display server of an HTML5 compatible browser.

### Requirements

A browser that supports HTML5 is required. Browsers that do not support HTML5 will display the swing trend chart regardless of the value of the `webChartFlag` property.

### Properties

A trend chart rendered in HTML5 supports the major properties available in trend graphs. However several minor properties are not supported or have limited support in HTML5-rendered trend charts, as follows:

| | |
|---|---|
| Background styles | Not supported. Except for the `bgColor` property, none of the background style properties are supported. This includes the properties whose names begin with `bg`, `traceBg`, and `border`. |
| Gradients | Gradient fill is not supported, so none of the properties named `*GradientStyle`, `*GradientMode`, or `*GradientFlag` are supported. |
| `markDefaultSize`<br>`markScale` | Not supported. These default to `6` and `No Scale`. The markers for each trace can be configured with the `traceNMarkStyle/Color` properties. |
| `traceNMarkStyle` | The markers for `traceN` can be enabled by setting `traceNMarkStyle > 0` but the shape of the markers (for example, circle, square, triangle) is selected automatically. |

| | |
|---|---|
| traceNType | Supported types are Line and Bar. Event type is not supported. |
| legendWidthPercent<br>legendValueMinSpace | Not supported. The legend (if visible) is sized automatically. |
| scrollbarMode/Size<br>zoomEnabledFlag | Not configurable. The scroll and zoom features are always enabled and the scrollbar size is fixed. |
| traceFillStyle | Supported values are None and Transparent. All other values (Solid, Gradient, Transparent Gradient) are converted to Transparent |
| x/yAxisMajor/<br>MinorDivisions | Not supported. The number of ticks on the x & y axis are selected automatically according to the size of the chart. |
| yAxisPostion | Not configurable. The y axis position is always outer-left. |
| yAxisAutoScaleVisTracesOnlyFlag | Not configurable, always true. |
| x/yAxisFlag x/<br>yAxisThickness | Not configurable. The x and y axes are always visible with a thickness of 1 and 2 pixels, respectively. |
| traceGroupNBandedFlag | Not supported. Trace groups are supported but banding within groups is not. |
| alert properties<br>(valueHighAlarm*,<br>valueLowAlarm*,<br>valueHighWarning*,<br>valueLowWarning*) | An HTML5-rendered trend chart supports one alert threshold per chart. If more than one alert threshold is enabled, HTML5 rendering will not be used. Also, the alert TraceColor is used only if the traceFillStyle is None. For any other traceFillStyle the alert TraceColor is ignored. The alert TraceStyle, Mark, and MarkColor are ignored. The alarmGlowFlag is not supported. |

The properties marked as not supported or not configurable in the table above will not appear in the dashboard builder's property sheet when a trend graph object is

selected and webChartFlag is checked. In some cases, if the properties listed above have been configured on a trend graph instance, then the webChartFlag property will automatically be set to false and hidden in the property sheet, because HTML5 rendering does not support those features. This occurs if any traceGroupNBandedFlag is checked, if more than one alarm threshold is enabled, or if the alarmGlowFlag is checked.

Rendering a trend chart in HTML5 supports two additional properties that are visible in the property sheet only if webChartFlag is checked:

| | |
|---|---|
| webChartNavigatorTrace | If this property is set to a trace number (a value between 1 and the traceCount value) the HTML5-rendered trend chart will display a "time navigator" at the bottom of the chart, just below the x (time) axis. The navigator plots the data for the indicated trace, and highlights the time range that is currently visible in the chart. The highlighted section can be resized or dragged to perform a time zoom or scroll. The navigator is intended to show the user the entire data set and let the user zoom/pan to the time range of interest. By default webChartNavigatorTrace = 0 so the navigator is disabled. |
| yAxisAutoScaleVisDataOnlyFlag | If this property is checked the chart will compute the y axis scale according to the min and max y values of the visible data points only. This means that the y axis scale may change as the user changes the visible time range by scrolling or zooming. By default the property is unchecked and the y axis is scaled according to the y values of all of the data points, visible or not. This matches the behavior of the trend graphs. Dashboard builder does not allow the webChartFlag property to be attached to data. This is by design, since the flag's value is expected to be constant. |

### Behavior

In addition to the properties listed above, there are some behavioral differences between an HTML5-rendered trend chart and a swing trend graph. These are described below:

- Legend

The legend does not show trace point data (`y`) values. Data values are only shown in the mouseover tooltips (if `cursorFlag = 1`). The cursor always snaps to the nearest data point, it does not show interpolated values between data points. Trace labels longer than 200 pixels are wrapped in the legend, and labels longer than 150 pixels are clipped in the tooltip.

- Y axis autoscaling

  Given the same `y` data values, an HTML5-rendered trend chart may choose a different value range for the `y` axes in autoscale mode as compared to swing trend graphs. This is because somewhat different algorithms are used.

- Reset button

  If the user changes the chart's visible time range, via the scrollbar or navigator or by dragging the cursor to perform a zoom, then a button labeled **Reset** will appear in the upper left corner of the plot area. A click on this button will reset the time axis to its original settings. The chart will also resume shifting to show the newest trace points, unless the `timeRangeBegin`/`End` properties are set.

- Data point grouping

  Rendering a trend chart in HTML5 makes use of a feature known as data grouping to enhance performance when a trace has many data points. With data grouping, the chart plots a single point using the average `y` value in cases where otherwise multiple points would be plotted on the same `x` pixel coordinate.

  When data grouping is in effect, the chart's tooltip will display a start time and end time (rather than the usual single time value) to indicate the time range of the averaged data points, for example: `05-Mar 12:35:00 ~ 05-Mar 12:45:00`.

  Notes: The data point grouping feature is enabled automatically and is not configurable. Also data grouping is performed independently of (and possibly in addition to) any data condensing or compaction that has already been applied by the historian. Also the `maxPointsPerTrace` property (`3200` in the `test1` display) is applied to the raw data, before any data grouping is applied.

  The `legendTimeFormat` property is used to format the date/time strings in the tooltip. If that property is blank then the `timeFormat` property is used instead. If the string contains a newline it is replaced with a space character to avoid making the tooltip overly tall.

- timeShift

  Rendering a trend chart in HTML5 does not attempt to keep the tick marks on the time axis aligned on even multiples of the `timeShift` value, in the case where `timeShift > 1`.

  Also, even if `webCharFlag` is set on all instances, the swing trend graph will still be used if a display is opened in Internet Explorer 8 or older, or if certain properties not supported by an HTML5-rendered trend chart (as listed above) are configured on a specific trend graph. To disable rendering trend charts in HTML5 globally, regardless of the `webChartFlag` value on individual objects, specify the

```
--apama.extendedArgs "-nohtml5"
```

argument on the display server command line.

#### Known issues and limitations

Consider the following when you use HTML5-rendered trend charts:

- After zooming or scrolling, the time axis labels may briefly be misaligned or overlap. They should be drawn correctly on the next refresh.
- The chart's tooltip may overlap the **Reset** button making it difficult to click the button. Moving the mouse a bit will correct this problem.
- HTML5 rendering will display all timestamps using the local time zone. This may cause user confusion if the display server is in a different time zone.
- Performance is affected by the number of traces, trace points, use of trace line shadows, trace line thickness, trace markers, trace fill, and other properties. Performance is also affected by the browser and version, and the CPU speed of the client host system.
- If any of the chart's graphical properties are changed while the chart is displayed (for example the `traceFillStyle` property is toggled by means of a checkbox control) the chart is rebuilt in the thin client, which in turn resets the chart's time range (as though the **Reset** button was clicked).
- Scrolling: If the mouse is moved below the bottom of the chart while dragging the scrollbar, the scrolling will stop. This is unlike the behavior of other objects, which will continue to scroll until the mouse button is released.
- Drilldown from trace points: If `traceFillStyle` is not `None` (that is, trace fill is enabled) and multiple traces share the same `Y` axis, then it is not possible to click on a point belonging to `traceN` if the fill area of a higher numbered trace is drawn over that point.
- When `yAxisLogFlag = 1` and the chart has multiple `y` axes and traces, some `y` axis labels may appear rather than numeric values. This is rare.
- Since HTML5-rendered trend charts do not support `legendWidthPercent`, the width of each chart's legend will vary according to the trace labels. This makes it difficult to create multiple trend chart instances on the same display whose time axes are all the same length, even if the charts all have the same width. (A single chart in stripchart mode may be more appropriate in those cases).
- On a touch interface, a swipe will scroll the chart left or right. To move the cursor without scrolling, tap the location to which you want the cursor to move.
- On a touch interface, a pinch-open gesture in the plot area, scrollbar, or navigator will zoom the chart's range in to the pinched range. A pinch-close gesture will zoom out to the pinched range. The pinch-close operation may be difficult to use. A tap on the **Reset** button gives a better zoom-out experience.

#### Hidden properties when webChartFlag is selected

Following is the list of `obj_trendgraph02` properties that are hidden when `webChartFlag` is checked:

- `borderPixels`
- `labelMinTabWidth`
- `legendWidthPercent`
- `legendBgGradientMode`
- `legendBgGradientColor2`
- `legendValueMinSpace`
- `markDefaultSize`
- `markScaleMode`
- `outlineColor`
- `scrollbarMode`
- `scrollbarSize`
- `zoomEnabledFlag`
- `timeRangeOfHistory`
- `traceBgColor`
- `traceBgGradientMode`
- `traceBgGradientColor2`
- `traceBgImage`
- `yAxisAutoScaleVisTracesOnlyFlag`
- `yAxisFormat`
- `yAxisFlag`
- `yAxisPosition`
- `yAxisValueLabels`
- `yAxisMajor/MinorDivisions`
- `xAxisFlag`
- `xAxisMajorDivisions`

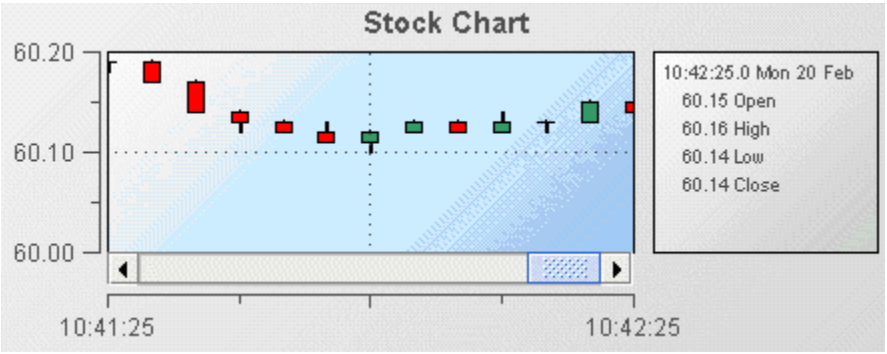
For each `traceN`, the following properties are hidden:

- `traceNValueAlarmStatus`
- `traceNValueAlarmStatusTable`

- `traceNValueHistoryFlag`
- `traceGroupNBandedFlag`
- `traceNYAxisGridVisFlag`
- `traceNYAxisMinLabelWidth`
- `traceNYAxisValueLabels`
- `traceNYAxisVisFlag`
- `traceNYAxisAutoScaleMode`
- `traceNYAxisValueMin`
- `traceNYAxisValueMax`

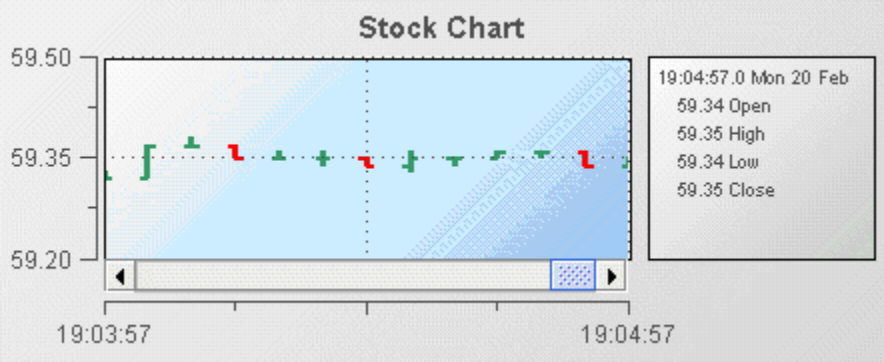
## Using stock charts

Stock charts provide the ability to view the Open, High, Low, and Close values (OHLC) for a scenario variable or DataView field such as stock price over set time intervals. The intervals may be small such as 5 seconds if being used for intra-day trading or larger for longer time periods such as hours, days, or weeks. The following illustration is from the Stock Chart tutorial sample.




In this example, the OHLC values are shown as a candlestick chart where each "stick" represents a 5 second interval. The stock chart supports others display formats such as OHLC, line, and bar.

1. Open the file `tutorial-stock-chart.rtv` by double-clicking **Stock Chart** on the tutorial main page.
2. Select the stock chart object and in the Object Properties panel, select the `priceTraceType` property and change its value to `OHLC`.




The data displayed is the same as that displayed in the previous illustration where `priceTraceType` was set to Candlestick. Only the presentation has changed.

Although named “stock chart” you are not limited to using it for stock data. You can chart Open, High, Low, and Close values for any scenario variable or DataView field. Other financial and non financial data can often benefit from being visualized as a stock chart.

Detailed reference information on stock charts is provided in "Stock charts" on page 472.

## Using OHLC values

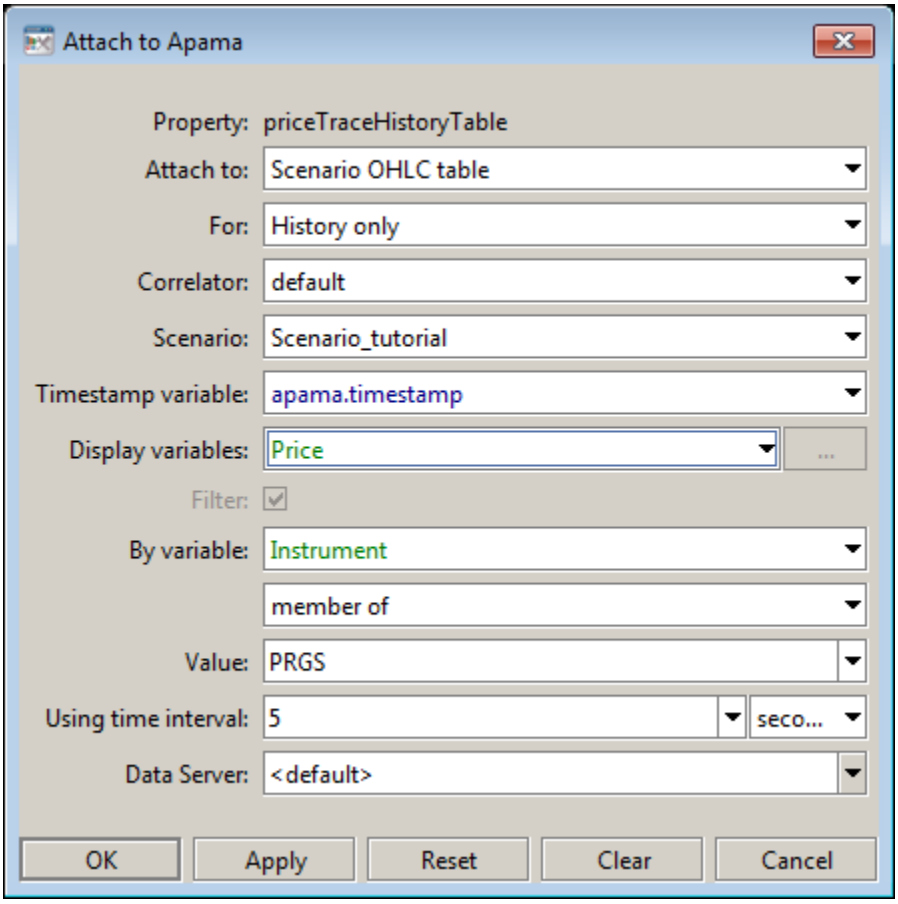
The OHLC values for a stock chart can be provided by attaching the stock chart to one of the following:

- OHLC table
- Scenario trend table (requires that the scenario have open, high, low, and close variables)
- Scenario instance table (requires that the scenario have open, high, low, and close variables)

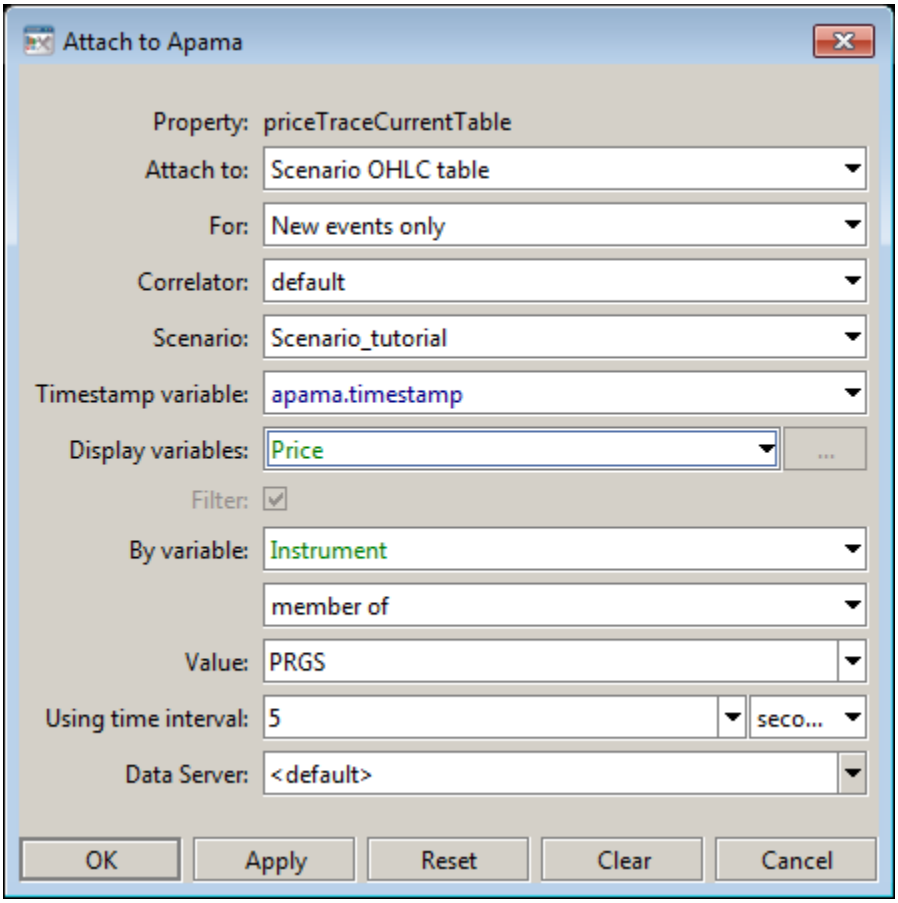
The simplest is to attach the chart to a scenario OHLC table. This is specified when creating the attachment in the Attach to Apama dialog.

When attaching to a scenario OHLC table, you need only specify the scenario variable you want to chart OHLC values for and a time interval. Apama will then automatically calculate the OHLC values. No modifications to your scenario are required. The following section uses the Stock Chart tutorial sample.

1. Select the stock chart object in the `tutorial-stock-chart.rtv` file.
2. In the Object Properties panel, double-click the `priceTraceHistoryTable` property to display the attachment settings for the stock chart




3. In the Object Properties panel, double-click the `priceTraceCurrentTable` property to display the attachment settings for the stock chart.

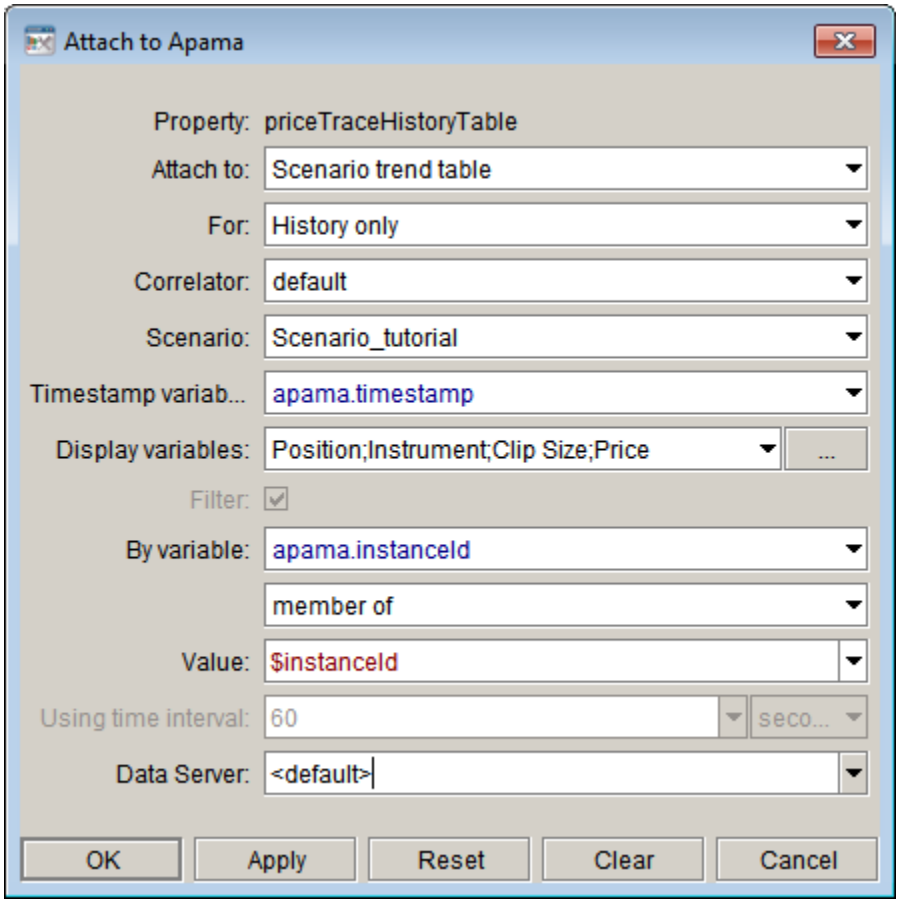


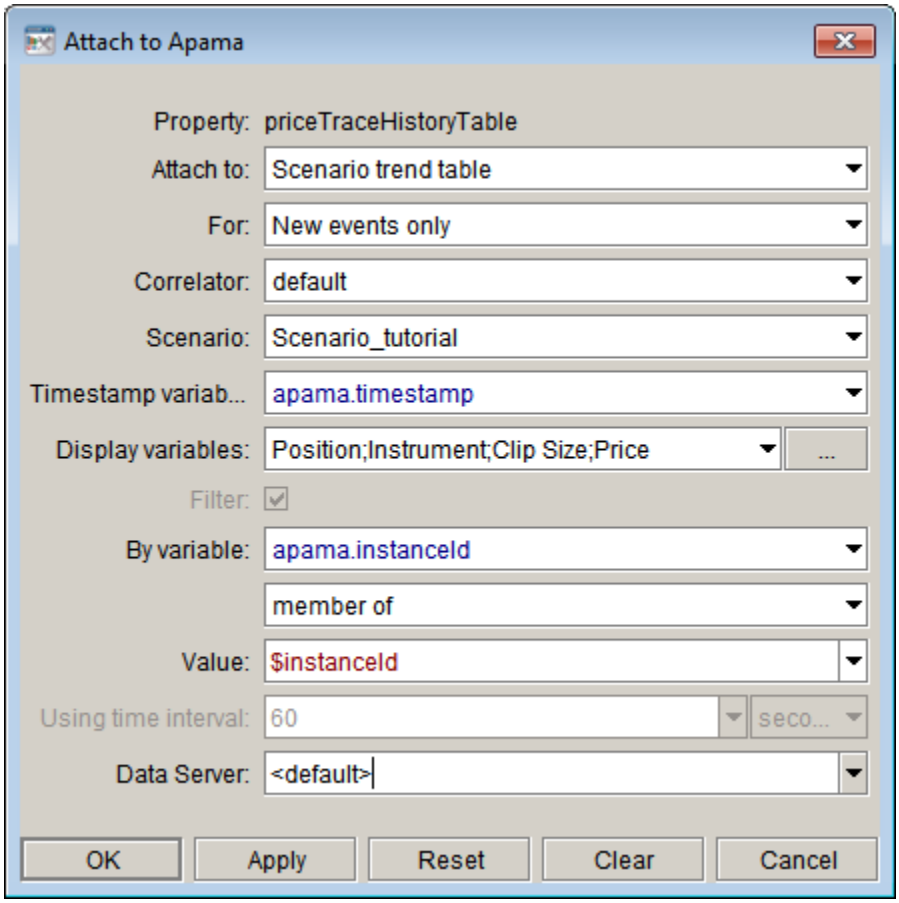

Here the attachment is made to the scenario OHLC table of the tutorial scenario and the `Price` variable is being displayed. This is the variable for which OHLC values will be calculated and displayed. The event timestamp, `apama.timestamp`, is the timestamp used to determine the time of events. The time interval is set to 5 seconds resulting in an OHLC value being charted every 5 seconds where each represents the preceding 5 seconds. The filter is set to match the scenario instance where the variable Instrument equals `APMA`.

When attaching to a scenario OHLC table, you must specify the time interval so that Apama knows what interval to use to calculate OHLC values. The time interval field is only enabled when attaching to a scenario OHLC table.

You must also specify a filter. As with trend charts a stock chart displays the value of one variable of a single scenario instance over time. If a filter matches more than one scenario instance the first found will be displayed.

The second way to provide OHLC data for a stock chart is to attach it to a scenario trend table. Do this when you want control of the calculation of OHLC values in a scenario. This requires that the scenario have variables for open, high, low, and close. When attaching a stock chart to scenario trend data you must specify for the display variables the individual open, high, low, and close variables of the scenario.


Attach to Apama
Property: priceTraceHistoryTable
Attach to: Scenario trend table
For: History only
Correlator: default
Scenario: Scenario_tutorial
Timestamp variab... apama.timestamp
Display variables: Position;Instrument;Clip Size;Price
...
Filter:
By variable: apama.instanceId
member of
Value: $instanceId
Using time interval: 60
seco...
Data Server: <default>
OK
Apply
Reset
Clear
Cancel

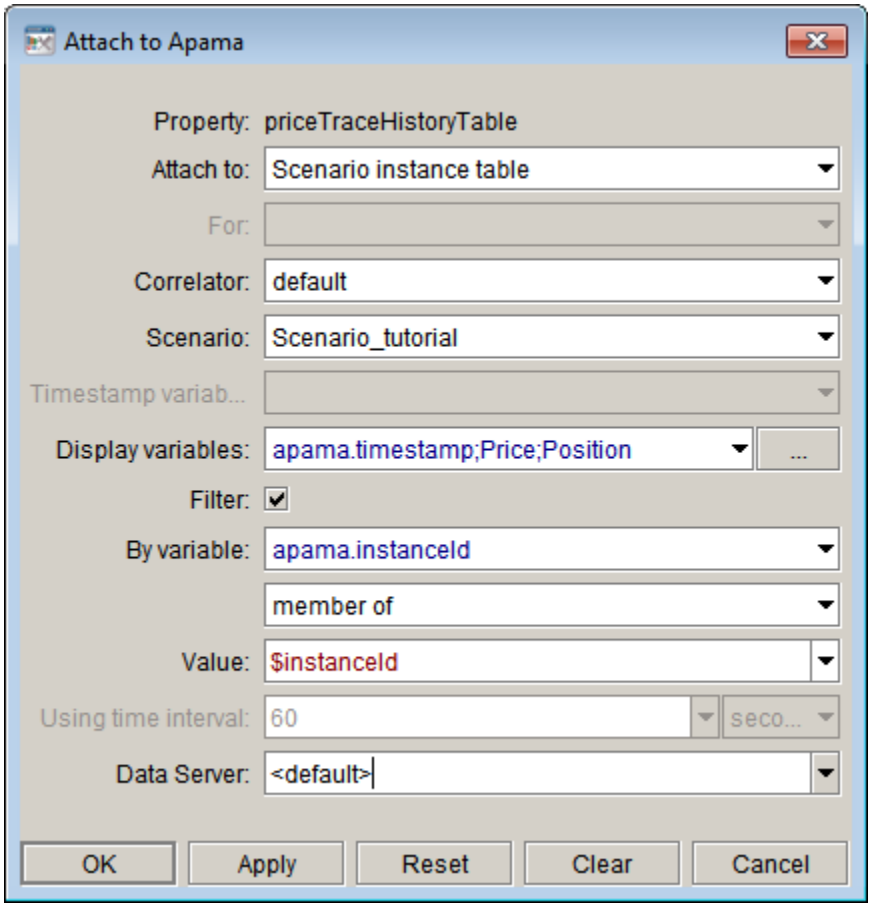



In this illustration, the attachment is made to the scenario trend table of the OHLC scenario. The scenario variables open, high, low, and close are used to provide the OHLC values. Notice that the **Using time interval** field is disabled. This is because the scenario is calculating the OHLC values; not the dashboard or dashboard server.

The names of the scenario variables do not matter. However they must be specified in the order open, high, low, and close. Only number variables can be used. String variables must be converted to numbers for use in stock charts.

The third way to provide OHLC data for a stock chart is to attach it to a scenario instance table. This is similar to attaching to a scenario trend table in that the scenario has control over the calculation of the OHLC values. It differs in that OHLC data for only one instance of the scenario is maintained in memory. This is valuable when you want to minimize memory use. However, it results in the chart being reset, cleared of all data, whenever OHLC values for a different scenario instance are displayed.

Use the `priceTraceHistoryTable` when attaching a stock chart to a scenario instance table. Attaching the `priceTraceCurrentTable` property to a scenario instance table will result in only the latest data value being displayed.



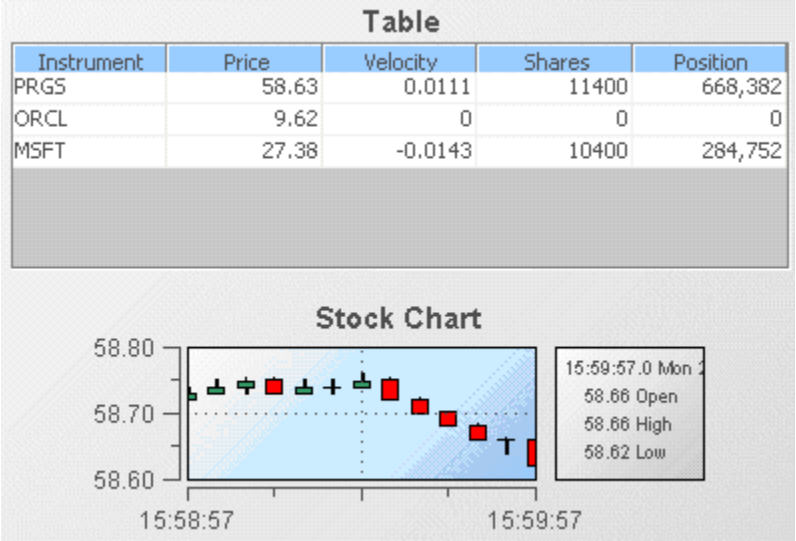

In this illustration, the attachment is made to the scenario instance table of the OHLC scenario. The scenario variables open, high, low, and close are used to provide the OHLC values. If you do not enable the `Timestamp variable` field for scenario instance table attachments, you need to specify the timestamp as the first entry in the `Display variables` field; here `apama.timestamp` is being used.

*Note:* Unless you have severe memory constraints or are displaying OHLC values for only a single scenario instance, you should attach the `priceTraceHistoryTable` property to either a scenario OHLC table or a scenario trend table, as this provides the best usage experience for the dashboard user.

## Creating a scenario stock chart

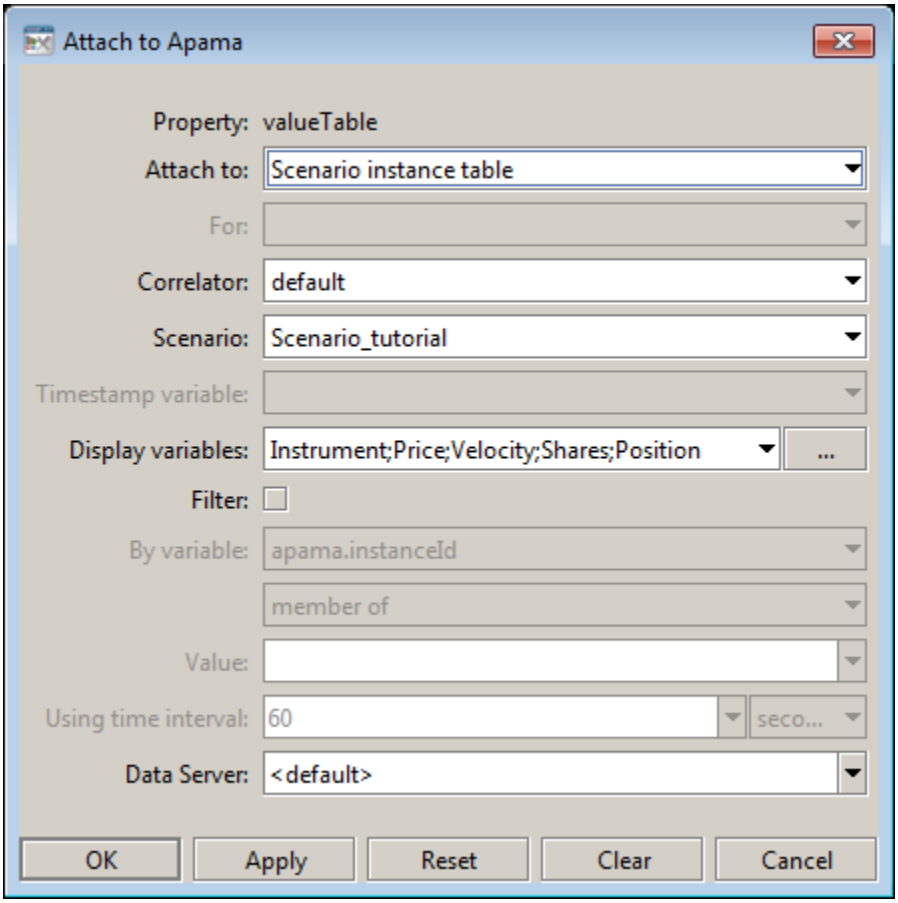
To create a stock chart for a scenario, you add it to a dashboard and attach its `priceTraceHistoryTable` property to OHLC data for a scenario instance. The filter to identify the instance will typically match on `$instanceId` although other filters could also be used.

The Stock Chart Drilldown tutorial sample demonstrates how to use a stock chart where the scenario instance charted is determined by the selection in a scenario summary table. The following illustration shows the stock chart from the Stock Chart Drilldown sample.



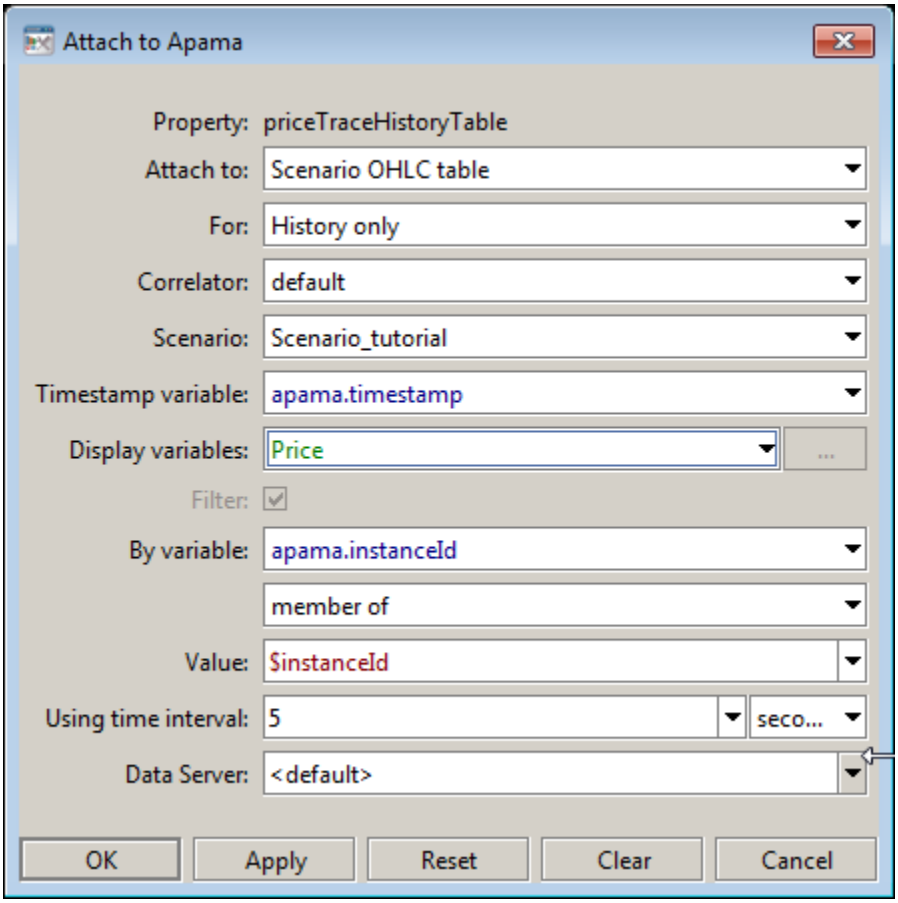

**To recreate this sample create a new dashboard and perform the following steps**

1. From the **Table** tab in the Object Palette, select the Table object and add it to the dashboard canvas.
2. With the table object selected, in the Object Properties panel double-click the table object's `valueTable` property and attach it to Apama and select the tutorial scenario. Select the display variables shown in the example and do not apply a filter. The information should be specified as follows:

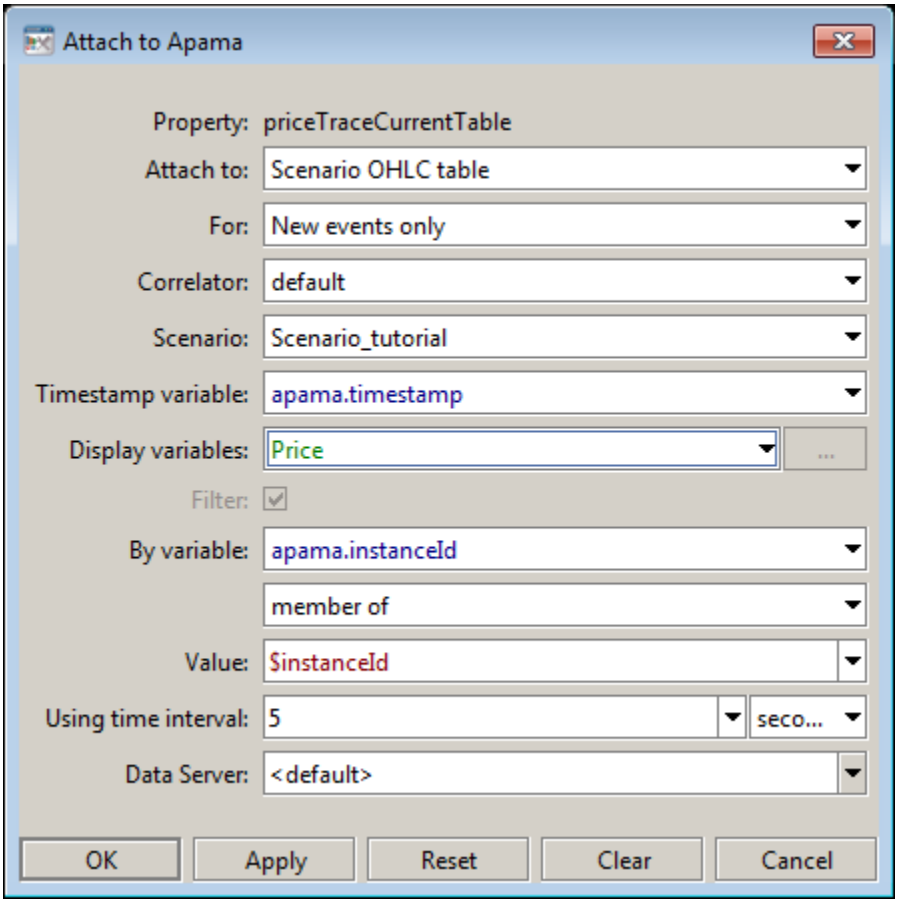



3. From the **Trends** tab in the Object Palette, select the Stock Chart object and add it to the dashboard Canvas.

4. With the stock chart selected, in the Object Properties panel, double-click the `priceTraceHistoryTable` property. Attach it to the OHLC table for the tutorial scenario and specify the rest of the information as shown in the following illustration. Here the `Price` variable will be charted over 5 second intervals. The scenario instance charted will be the selected instance as indicated by the variable $instanceId




5. With the stock chart selected, in the Object Properties panel, double-click the `priceTraceCurrentTable` property. Attach it to the OHLC table for the tutorial scenario and specify the rest of the information as shown in the following illustration. Here the `Price` variable will be charted over 5 second intervals. The scenario instance charted will be the selected instance as indicated by the variable $instanceId.

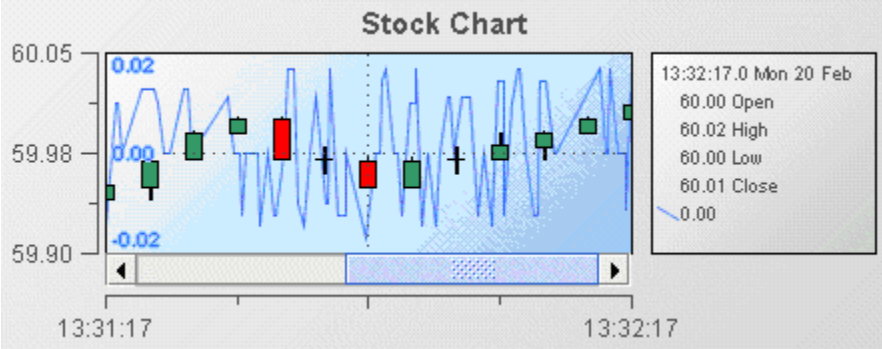



6. Select the `timeRange` property and set its value to `60.0`. This will set the chart's time axis such that 60 seconds of data will be visible. If you set the value too high you may encounter problems where the "sticks" of the chart are close and overlap.
7. Select the `scrollbarMode` property and change its value to `As Needed`. This will add a scrollbar to the chart allowing you to scroll back in time to view earlier values.
8. Select a scenario instance in the table by double-clicking on it. The chart will now begin charting OHLC values for the `Price` variable of the selected scenario instance.

If you have not previously displayed a sample containing a stock chart, values in the chart will not appear for ten seconds. Apama does not collect data in a scenario OHLC table until the first attachment to an instance of the table is made.

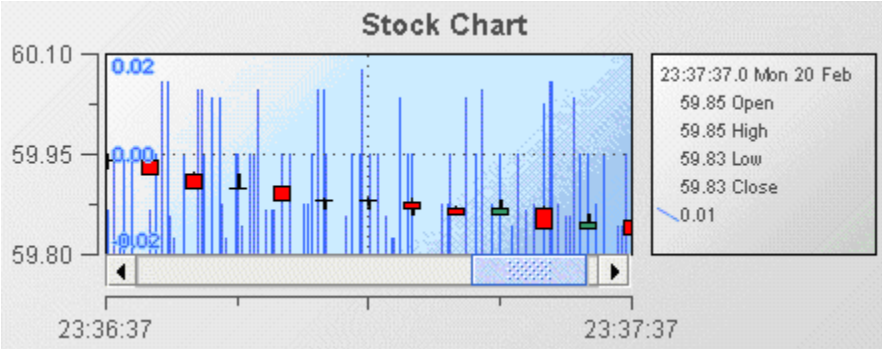
## Adding overlays

Stock charts support up to nine overlays. An overlay is much like a trace in a trend chart. Overlays can be used to compare the displayed OHLC values against other variables or fields such as other stock prices, overall activity on the stock index, or to show periodic events such as stock splits and earnings announcements. The following illustration is from the Stock Chart Overlays tutorial sample.



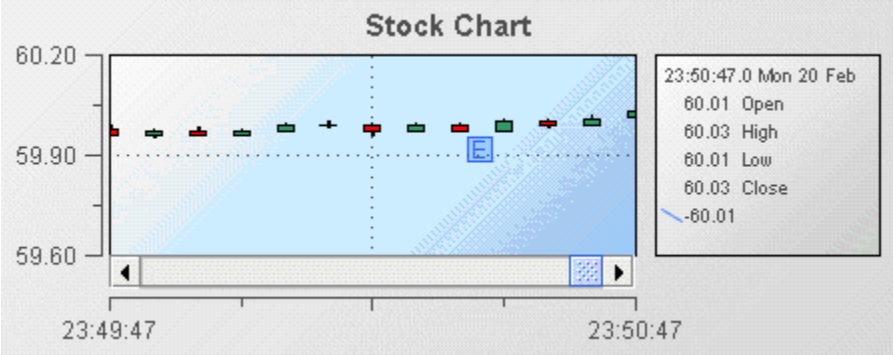

Here the overlay is showing the velocity of the stock price. Notice that multiple scales are shown on the Y axis; the outer scale corresponds to the stock price and the inner scale the velocity.

1. Open the file `tutorial-stock-overlay.rtv` by selecting **Stock Chart Overlays** on the tutorial main page.
2. Select the stock chart and in the Object Properties panel, select the property `overlay1Type` and change its value to `Bar`.




The overlay values are now displayed as discrete bars and not as a single line.

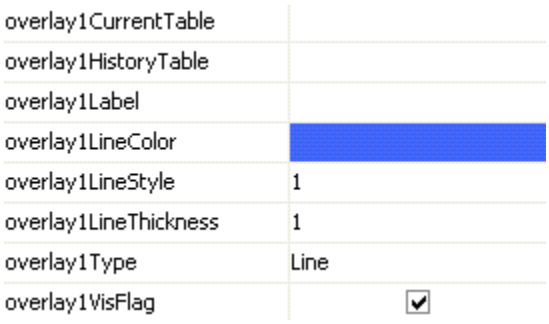
If you set `overlay1Type` to `Event`, event markers will be placed on the chart at the occurrence of each event. This allows you to easily identify when key events occurred. The following illustration demonstrates this.




The character displayed in event markers is the first letter of the corresponding `overlayNLabel` property.

When using overlays to display event markers, the event markers should be relatively sparse. Displaying high numbers of event markers will cause them to overlap and limit their usefulness.

To add overlays to a stock chart, set the `overlayCount` property to the number of overlays to be displayed. This will cause a set of properties to be added to the property panel for each overlay; overlay1 through overlayN. Following are the properties for overlay 1.



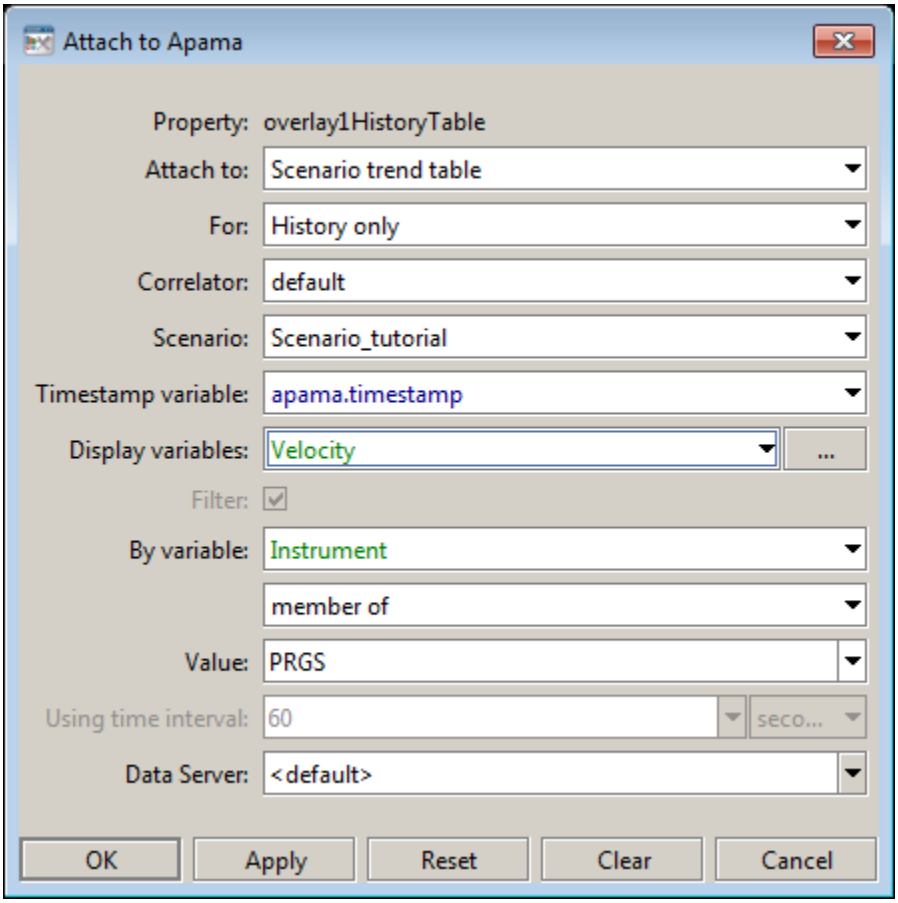

When you attach an overlay to a scenario OHLC table or a scenario trend table, use the properties `Overlay`*n* `CurrentTable` and `Overlay`*n* `HistoryTable`.

Use only the `Overlay`*n* `CurrentTable` property when attaching the overlay to a scenario instance table. Attaching to a scenario instance table requires less memory but the resulting overlay may be missing one or more data points. This can occur if the dashboard is running on a heavily loaded system. Unless you have severe memory restrictions, you should not attach overlays to a scenario instance table. Better results can be achieved by attaching to a scenario trend table. This will guarantee that the overlay contains all data points.

### *Recreating the Stock Chart Overlay sample*

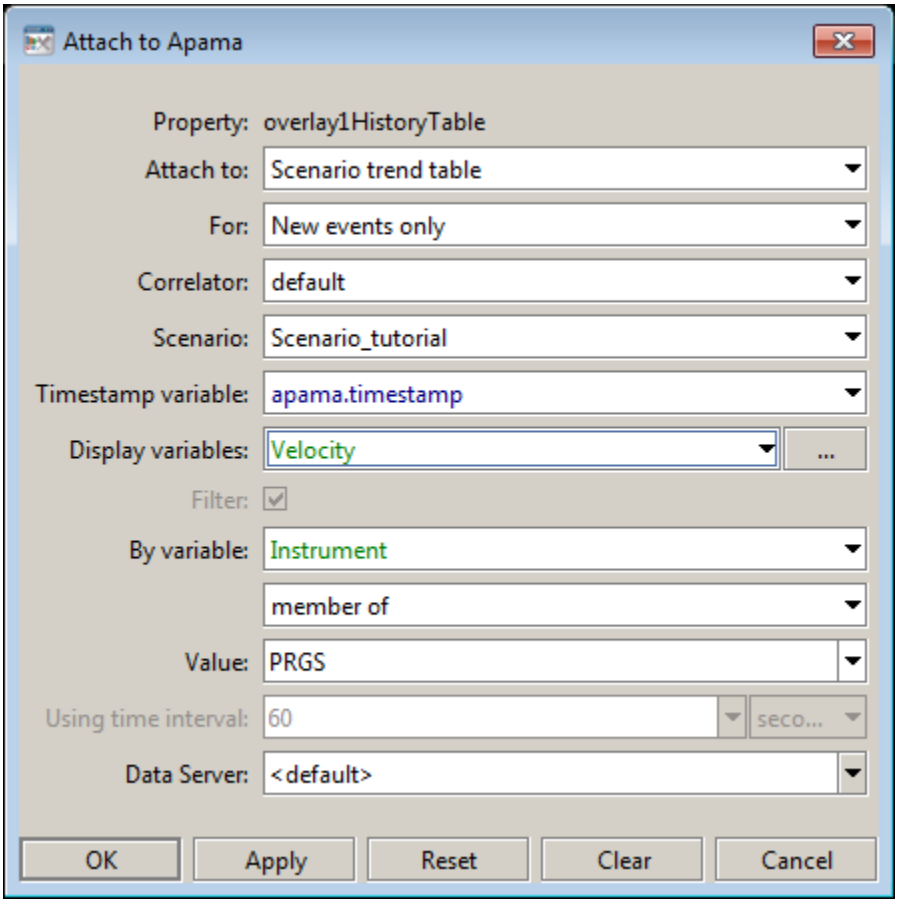
**To recreate the Stock Chart Overlay sample**

1. Open the file `tutorial-stock-chart.rtv` by selecting **Stock Chart** on the tutorial main page.
2. Select the stock chart and in the Object Properties panel, select the property `overlayCount` and change its value to `1`. This will cause the `overlay1` properties to be added to the property panel.
3. Double-click the `overlay1HistoryTable` property and attach it to a scenario OHLC table by specifying the following information.



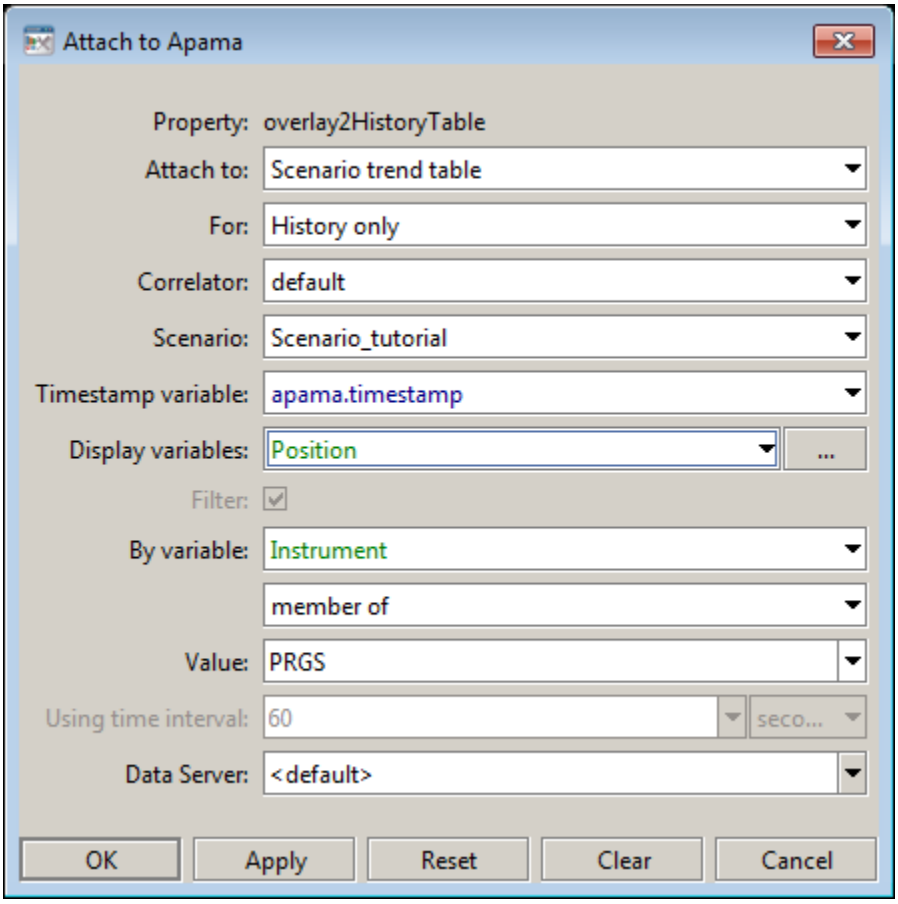

The overlay is now attached to `Velocity` property of the scenario instance where the `Instrument` equals `APMA`; this is the same filter used for the `priceTraceHistoryTable` attachment.

4. Double-click the `overlay1CurrentTable` property and attach it to a scenario OHLC table by specifying the following information.

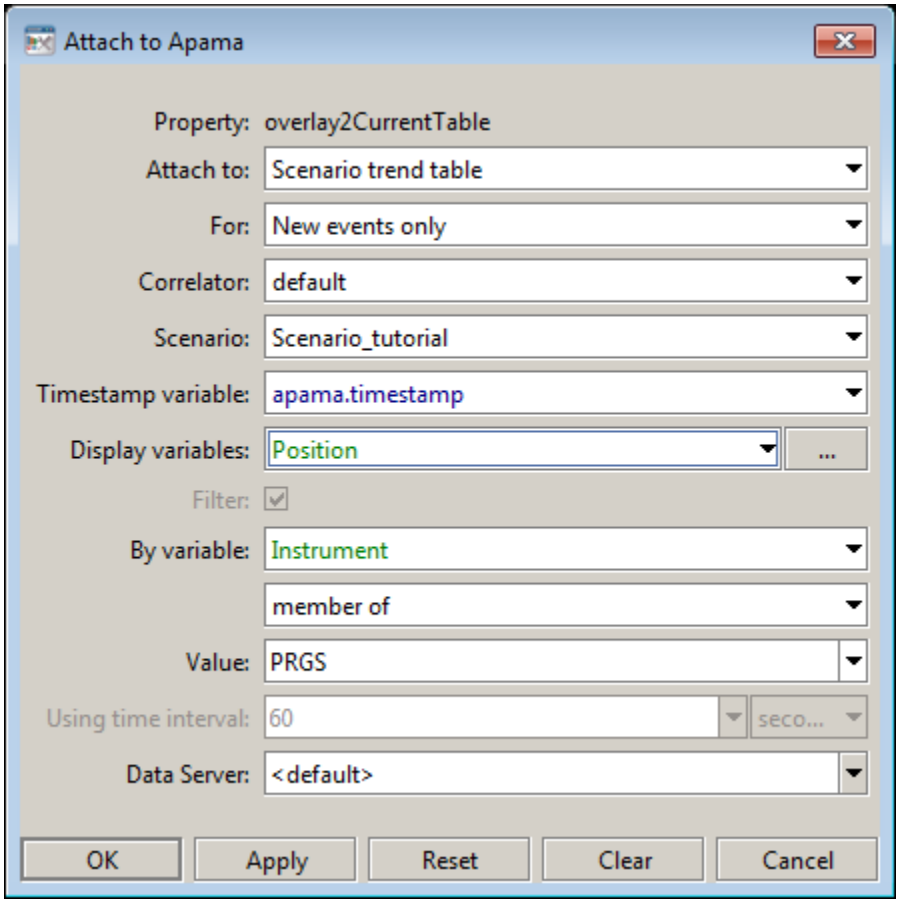



The overlay is now attached to `Velocity` property of the scenario instance where the `Instrument` equals `APMA`; this is the same filter used for the `priceTraceHistoryTable` attachment.

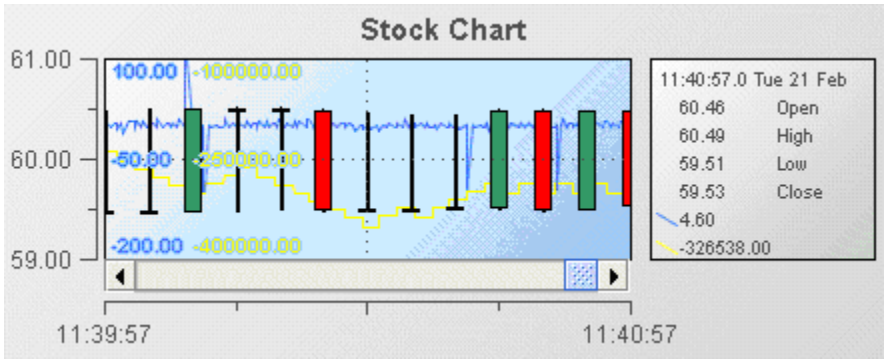
5. Select the `overlayCount` property in the property panel and change its value to `2`. This will cause the `overlay2` properties to be added to the property panel.
6. Double-click the `overlay2HistoryTable` property and attach it to a scenario trend table by specifying the following information:




7. Double-click the `overlay2CurrentTable` property and attach it to a scenario trend table by specifying the following information:




The stock chart now contains two overlays; one showing the velocity of the stock price and the second showing the current position in that instrument. The following illustration shows how this looks in the sample.




Overlays can be hidden by tuning off the `overlayNVisFlag` property. This is for use when building dashboards where you will have input controls such as checkboxes which will allow the user to hide or show different overlays.

## Generating OHLC values

If you generate OHLC values, you should also use a scenario variable or DataView field as the timestamp. If you use `apama.timestamp`, you need to design the scenario or DataView to generate update events only when the OHLC values change. Your dashboard will add an OHLC data point to a Stock Chart for every Update event it receives. If a scenario, for example, generates Update events in response to other variables changing and `apama.timestamp` is being used as the timestamp then spurious OHLC data points will be added to the chart. If the chart were displaying a candlestick this would manifest itself as extra "sticks" appearing in the chart.

If you use a scenario variable or DataView field as the timestamp, data points will only be added to the chart when timestamp and/or OHLC values have changed.

Furthermore, the update of the OHLC values must occur as a whole; that is each Update event must contain the updated value of each of the Open, High, Low, and Close variables. If the update of each variable were to generate a separate Update event, you would also have spurious data points in the chart. This is because your dashboard has no way of knowing if the unchanged values are correct or not.

To update the OHLC variables in a single update event your scenario or DataView needs to set the value of each in the scope of a single rule. Following is an example of this in an Event Modeler rule.

Here local variables `_open`, `_high`, `_low`, and `_close` are used throughout the scenario to calculate the OHLC values. Within this rule the output variables `open`, `high`, `low`, and `close` are being set to these values such that a single Update event contains the updated value of each.

If you use a scenario variable or DataView field as the timestamp, data points will only be added to the chart when timestamp and/or OHLC values have changed.

Furthermore, the update of the OHLC values must occur as a whole; that is each Update event must contain the updated value of each of the Open, High, Low, and Close variables. If the update of each variable were to generate a separate Update event, you would also have spurious data points in the chart. This is because your dashboard has no way of knowing if the unchanged values are correct or not.

To update the OHLC variables in a single update event your scenario or DataView needs to set the value of each in the scope of a single rule. Following is an example of this in an Event Modeler rule.


Here local variables `_open`, `_high`, `_low`, and `_close` are used throughout the scenario to calculate the OHLC values. Within this rule the output variables `open`, `high`, `low`, and `close` are being set to these values such that a single Update event contains the updated value of each.

## Localizing dashboard labels

You can localize dashboard labels by attaching XML data (filtered based on the end-user-specified value of a dashboard variable) to the object properties that specify the labels. For a complete localization example, select **Localization** on the Dashboard Builder Tutorial main page.

### To localize dashboard labels

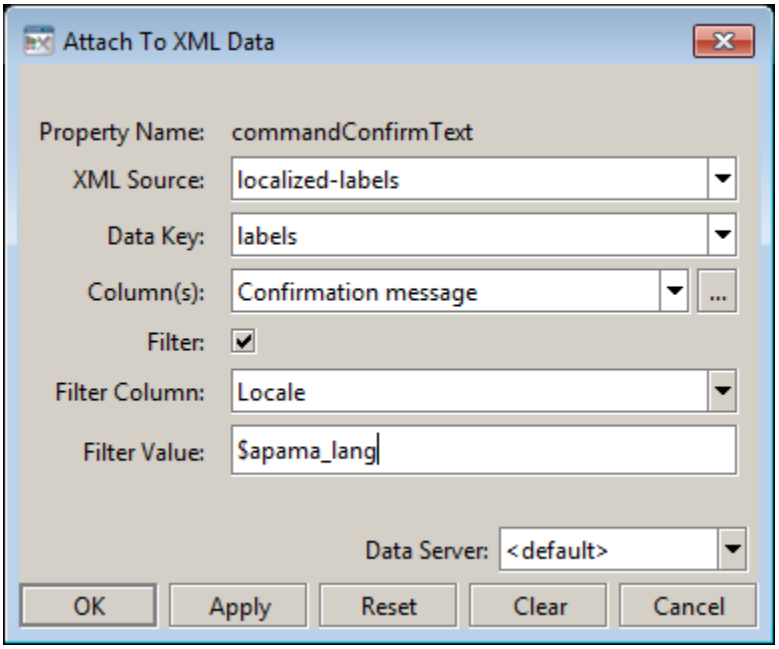
1. Create an XML dataset with a tabular data element. (See "Using XML Data" on page 235.) Create a column for supported locales, as well as a column for each label. Create a row for each locale. In each row, put a specific locale and the text for each label localized for that specific locale. Here is an example from the Builder tutorial:

```
<?xml version="1.0" encoding="UTF-8"?>
<dataset xmlns="www.sl.com" version="1.0">
<table key="labels">
 <tc name="Locale"/>
 <tc name="Confirmation message"/>
 <tc name="Press button"/>
 <tc name="Are you sure"/>
 <tc name="Numeric input"/>
 <tc name="Datetime input"/>
 <tc name="Numeric display"/>
 <tc name="Datetime display"/>
 <tr name="English">
  <td>en_US</td>
  <td>Confirmation message:</td>
  <td>Press button</td>
  <td>Are you sure?</td>
  <td>Numeric input:</td>
  <td>Date time input:</td>
  <td>Numeric display:</td>
  <td>Date time display:</td>
 </tr>
 <tr name="French">
  <td>fr_FR</td>
  <td>Message de confirmation:</td>
  <td>Bouton-poussoir</td>
  <td>Etes-vous sûr?</td>
```

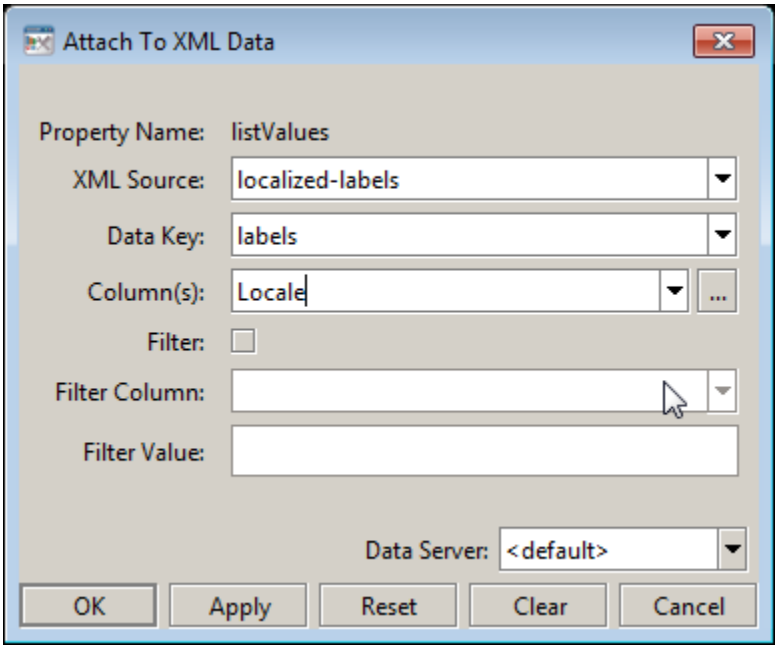
```
  <td>Entrée numérique:</td>
  <td>Entrée date-heure:</td>
  <td>Affichage numérique:</td>
  <td>Affichage date-heure:</td>
 </tr>
 <tr name="Spanish">
  <td>es_ES</td>
  <td>Mensaje de confirmación:</td>
  <td>Botón</td>
  <td>¿Estás seguro?</td>
  <td>Entrada numérica:</td>
  <td>Entrada de la fecha y hora:</td>
  <td>Exhibición numérica:</td>
  <td>Exhibición de la fecha y hora:</td>
 </tr>
 <tr name="Chinese">
  <td>zh_TW</td>
  <td>#### :</td>
  <td>##</td>
  <td>#####</td>
  <td>####:</td>
  <td>##-#####:</td>
  <td>####:</td>
  <td>##-#####:</td>
 </tr>
</table>
</dataset>
```

This file defines labels **Press button**, **Are you sure?**, and so forth, for the languages English, French, Spanish, and Chinese. The first column `Locale` defines the locale, or language, of the corresponding row.

2. For each object property that specifies a label, attach the property to the column that corresponds to that label, filtered to select the row for which the value in the locale column is the value of a dashboard variable that specifies the locale desired for the end user. You can use the predefined variable `$apama_lang` for this purpose. Here is an example:




3. Provide a way for end users to set the relevant variable (for example, the predefined dashboard variable `$apama_lang`) to their desired locale. One way to do this is to include, on your top-level dashboards, a combo box (from the **Controls** tab). Attach the `selectedValue` and `varToSet` properties of the combo box to `$apama_lang`, and attach the `listValues` property to the locale column of your XML data element. Here is an example:




The dashboard substitution `$apama_lang` is automatically defined for dashboards. Use ISO 639 language codes as values of this variable. This is the same locale string

used within Java. See the Java documentation for more information on locales within Java. Here are some sample locale values:

For dashboards in Builder and Viewer connected directly to the Correlator, the default value for `$apama_lang` is what Java reports as the locale in the `Locale` object as derived from the host system's locale.

| Locale Name | Locale |
|---|---|
| Locale.CHINA | zh_CN |
| Locale.CHINESE | zh |
| Locale.SIMPLIFIED_CHINESE | zh_CN |
| Locale.TRADITIONAL_CHINESE | zh_TW |
| Locale.PRC | zh_CN |
| Locale.TAIWAN | zh_TW |
| Locale.ENGLISH | en |
| Locale.UK | en_GB |
| Locale.US | en_US |
| Locale.FRANCE | fr_FR |
| Locale.FRENCH | fr |

For deployed dashboards, the value of `$apama_lang` is set based on the locale of the host on which the dashboard Display Server or Data Server is running. A single dashboard server cannot serve dashboards to users in different languages. Note that number and date formatting performed by the dashboard server are always controlled by the system locale.

**Note:** Numeric formats (1,000.00 versus 1.000,00) are controlled by the system locale. You cannot change this by setting `$apama_lang`. The only way to override it, other than changing your system locale, is through Java system properties. Date/time formats are also controlled by the system locale.

## Localizing dashboard messages

For thin-client (Display Server) deployments, you can localize the text displayed in popup menus, login windows, status windows, and various error messages.

### To localize dashboard messages

1. Extract the file `rtvdisplay_strings.properties` from the `WEB-INF/classes/gmsjsp` directory of the `rtvdisplay.war` file in your deployment package. Copy it to a new file with the desired locale suffix (for example, `rtvdisplay_strings_ja.properties` for Japanese).
2. Edit the new file so that it contains the localized text.
3. Pack the edited file into `rtvdisplay.war`, in `WEB-INF/classes/gmsjsp`.

   The locale setting of your application server is used to determine which properties file to load. If the application server does not have the desired locale setting for the thin client, edit the original file (`rtvdisplay_strings.properties`) and pack it into the `.war` file.

# 4 Using Dashboard Functions


- Using built-in functions ..... 136
- Creating custom functions ..... 139

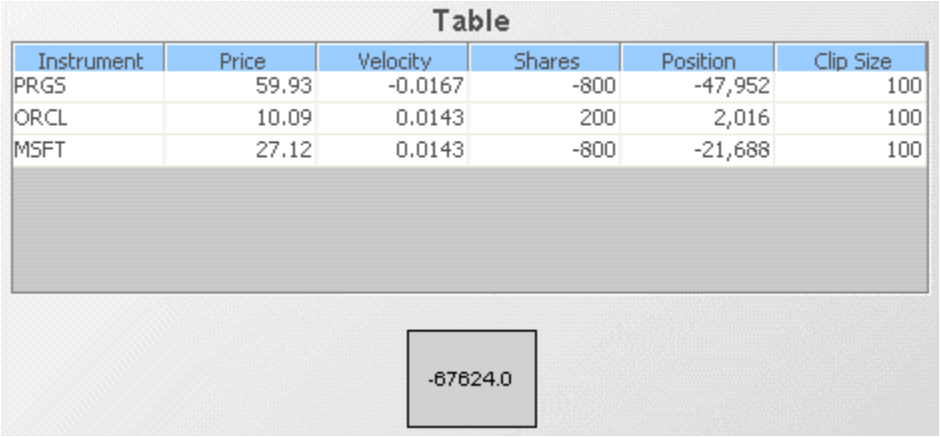
You can use Dashboard functions in order to perform calculations, filtering, formatting and other operations on correlator data. Scalar functions can be used when operating on a single variable of a single scenario instance. Tabular functions can be used when operating on a table of correlator data. Where all correlator data is stored in dashboards or dashboard servers as tables, they are compatible with all tabular functions.

## Using built-in functions

Following is an example of using a built-in function.

**To use a built-in function**

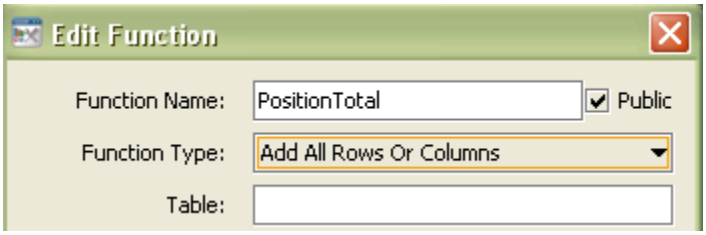
1. Open the file `tutorial-function-sum.rtv` by selecting **Data Functions** on the tutorial main page.



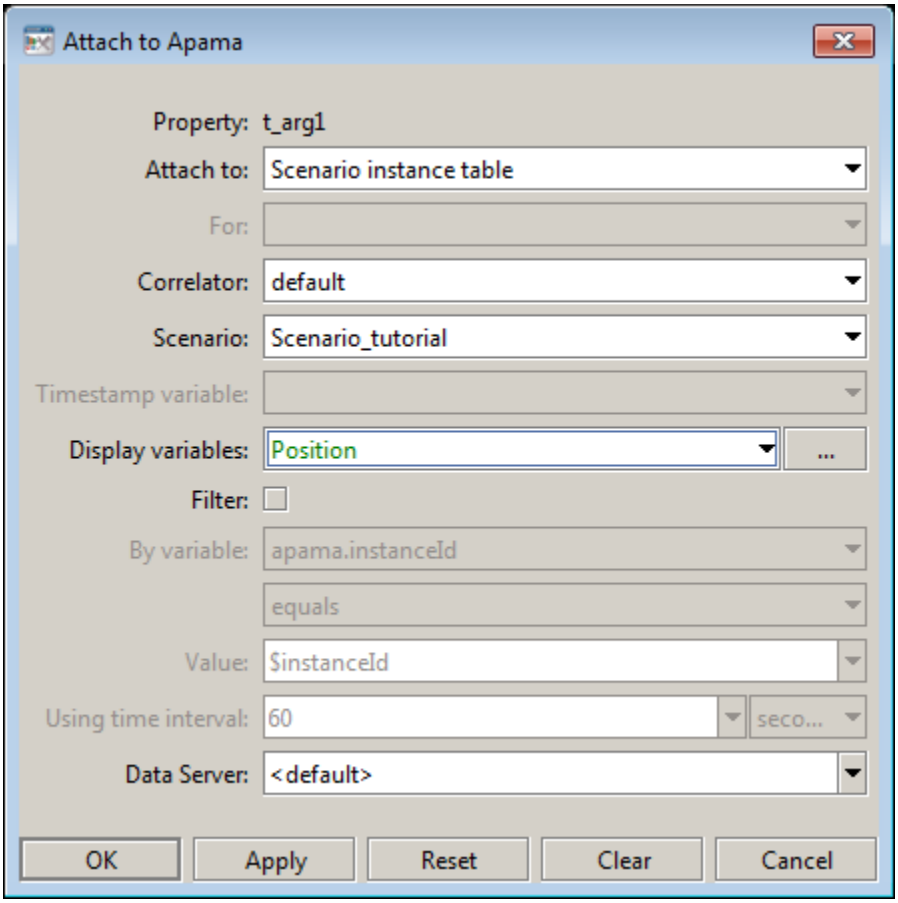

   Here the value in the label at the bottom of the dashboard is the sum of the `Position` variables of each scenario instance. To recreate this sample follow the following steps.

2. Open the file `tutorial-summary-table.rtv` by selecting **Summary Table** on the tutorial main page.
3. From the **Tools** menu select the **Functions** item to display the Functions panel.
4. Click the **Add** button to display the Edit Function dialog.
5. Set the **Function Name** field to `PositionTotal` and the **Function Type** field to `Add All Rows Or Columns`.

   For information on all built-in functions, see "Dashboard Function Reference" on page 541.




6. Right-click the **Table** field in the **Edit Function** dialog and attach it to Apama by specifying the information shown in the dialog shown below.




Here the attachment specifies that the `Position` column for the tutorial scenario is to be used. The **Sum** function will produce the sum of the values in all the cells of a given column; in this case the sum of the cells in the **Position** column for all instances of the scenario.

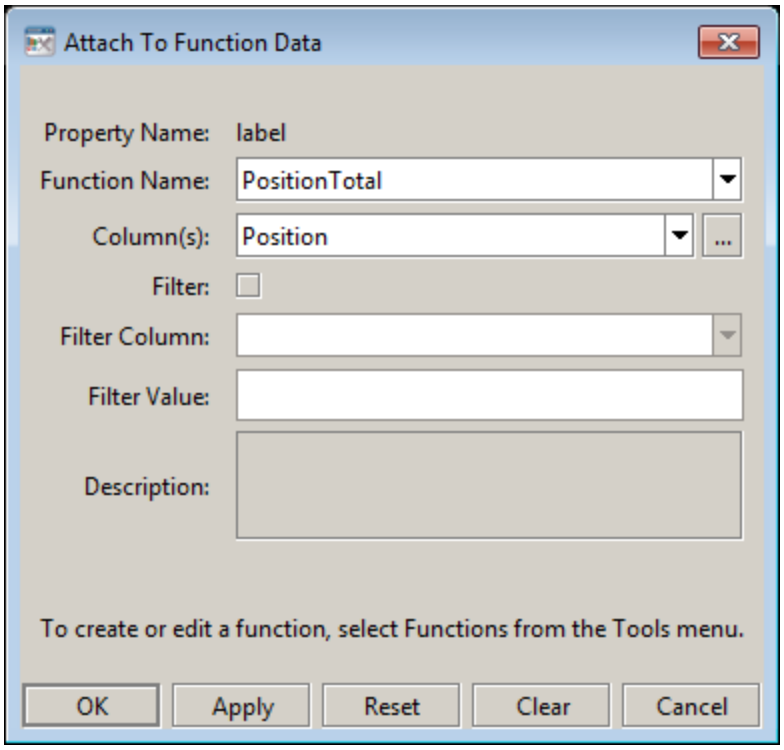
7. Click **OK** in the Edit Function dialog and close the Functions dialog.

   The function `PositionTotal` has now been defined and object properties can be attached to it.

8. From the **Labels** tab in the Object Palette, select the second label object and add it to the dashboard canvas.


9. Select the label object and in the Object Property panel right click the `valueString` property and select **Attach to Data | Function** to display the Attach to Function Data dialog.

10. In the Attach to Function Data dialog select the `PositionTotal` function as follows:




The label object is now attached to the `PositionTotal` function and will display the sum of the `Position` variable for all instances of the scenario.

For more information on the Functions panel and the Edit Function dialog, see "Introduction to Dashboard Functions" on page 543.

Dashboard Builder provides many functions for operating on data. These can be used to operate on scenario data to produce scalar results such as a sum. They can also be used to produce tabular results which can be displayed as tables or charts. It is also possible to chain functions where one function takes as its input value the output of another function. For more information, see the Dashboard Function Reference in *Developing Apama Applications*.

Apama also gives you the ability to define custom dashboard functions, as described in the next section.

# Creating custom functions

**To provide a library of functions**

1. Develop an implementation of `com.apama.dashboard.function.IFunctionLibrary`. See "Developing a custom-function library" on page 139.
2. Install your implementation. See "Installing a custom-function library" on page 140.

## Developing a custom-function library

A sample implementation of `IFunctionLibrary` is included below in "Sample IFunctionLibrary implementation" on page 141.

- Your implementation of `IFunctionLibrary` must implement the following methods:
- `getFunctionDescriptors`: Creates a function descriptor for each function that the library supports; returns a list of `com.apama.dashboard.function.IFunctionDescriptor`s. This method is called once at Data Server or Display Server startup. See "Implementing getFunctionDescriptors" on page 139.
- `evaluateFunction`: Returns the result of executing a specified function with specified arguments. See "Implementing evaluateFunction" on page 140.

When you compile your implementation, ensure that `ap-dashboard-client.jar` is on your class path. This `jar` file is in the `lib` directory of your Apama installation.

### *Implementing getFunctionDescriptors*

To create a function descriptor, use the factory class `com.apama.dashboard.function.FunctionDescritporFactory`. Call `createFunctionDescriptor`, passing arguments that specify the following:

- The function name that will be used by the Dashboard Builder and by the implementation of `evaluateFunction`
- The argument names that will be used by the Dashboard Builder
- The argument names that will be used by the implementation of `evaluateFunction`
- The return type of the function (`String`, `Double`, `Integer`, or `com.apama.dashboard.data.ITabularData`)
- The names of the returned columns, for functions that return table data
- A text description of the function

**Note:** When you create a dashboard custom function you must specify prefixes for parameters according to the parameter type. A prefix must be `s_arg` for a `String` parameter, `t_arg` for a `Table` parameter or `i_arg` for an `Integer`

parameter, for example, `s_arg1, s_arg2`. You can see sample code that shows this in the `getFunctionDescriptors()` definition near the beginning of "Sample IFunctionLibrary implementation" on page 141.

#### *Implementing evaluateFunction*

Implement this method to evaluate a specified function with specified actual arguments. The function is specified with the function name. The arguments are specified with an instance of `com.apama.dashboard.data.IVariableData`.

For functions that return table data, use the factory class `com.apama.dashboard.data.TabularDataFactory` to create an instance of `ITabularData`.

When you compile your implementation, ensure that `ap-dashboard-client.jar` is on your class path. This `jar` file is in the `lib` directory of your Apama installation.

Your implementation of `evaluateFunction` can set or retrieve substitution values, if necessary, by using the following methods of `DashboardManager` and `IDashboardContext`:

- `DashoardManager.getFunctionDashboardContext`: This static method takes as argument an instance of `IVariableData` and returns an instance of `IDashboardContext`. Pass the instance of `IVariableData` that is passed into `evaluateFunction`.
- `IDashboardContext.getSubstitution`: Gets the value of a substitution with a given name.
- `IDashboardContext.setSubstitution`: Sets the value of a substitution with a given name.
- `IDashboardContext.setSubstitutions`: Sets the values of substitutions, where the substitutions and values are specified with `String` vectors.

Each set method has a boolean argument, `triggerUpdate`, which controls whether objects attached to the substitution are updated. If it is `false`, they are not. If the substitutions are only used as command parameters or in drilldowns, you can improve performance by specifying `false`.

Here is an example:

```
IDashboardContext ctxt =
DashboardManager.getFunctionDashboardContext(v);
String val1 = ctx.getSubstitutionValue("$subst1");
...
ctx.setSubstitution("$subst2", "val2", false);
```

## Installing a custom-function library

To install your function library for a given Data Server or Display Server, do both of the following:

- Include a line in the Data Server or Display Server's `EXTENSIONS.ini` file that specifies the fully qualified name of your `IFunctionLibrary` implementation. The line must have the following form:

  ```
  function fully-qualified-classname
  ```

- create a `jar` file that contains your `IFunctionLibrary` implementation, and either add it to `APAMA_DASHBOARD_CLASSPATH` (changes to this environment variable are picked up by dashboard processes only at process startup) or add it to the list of **External Dependencies** in your project's **Dashboard Properties**. (In Software AG Designer, right-click your project and select **Properties**, expand **Apama**, select **Dashboard Properties**, activate the **External Dependencies** tab, and click the **Add External** button). You can also use the `--dashboardExtraJars` command line argument to specify this `jar` file.

A Data Server or Display Server's `EXTENSIONS.INI` is, by default, located in the `lib` directory of its Apama installation. You can specify a Data Server or Display Server's `EXTENSIONS.ini` file at startup by using the `-X` or `--extensionFile` option — see *Deploying Apama Applications*.

The `EXTENSIONS.ini` specifies the function library to use. This file identifies all the user supplied extension classes (including command libraries and scenario authorities). Here is a sample `EXTENSIONS.ini`:

```
function com.apama.dashboard.sample.SampleFunctionLibrary
command com.apama.dashboard.sample.SampleCommandLibrary
scenarioAuthority com.apama.dashboard.sample.SampleScenarioAuthority
```

This file installs a function library, a command library, and a scenario authority.

## Sample IFunctionLibrary implementation

Below is a sample implementation of `IFunctionLibrary`, which you can find under `samples\dashboard_studio\tutorial\src`:

```
package com.apama.dashboard.sample;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.StringTokenizer;
import java.util.Vector;
import java.util.regex.Pattern;
import com.apama.dashboard.data.ITabularData;
import com.apama.dashboard.data.IVariableData;
import com.apama.dashboard.data.TabularDataFactory;
import com.apama.dashboard.data.internal.TabularData;
import com.apama.dashboard.function.FunctionDescriptorFactory;
import com.apama.dashboard.function.IFunctionDescriptor;
import com.apama.dashboard.function.IFunctionLibrary;
/**
 * SampleFunctionLibrary is an example of a custom function library for
 * Dashboard Studio. Custom functions allow you to extend Dashboard Studio
 * by the additon of custom functions to process data for use as data
 * attachments.
 * <p>
 * SampleFunctionLibrary implements the functions:
 * <ul>
 * <li><b>String to Table</b>: Parses an encoded string to produce tabular
```

```
 * data.
 * </ul>
 *
 * $Copyright(c) 2013 Software AG, Darmstadt, Germany and/or its licensors$
 * @version      $Id$
 */
public class SampleFunctionLibrary implements IFunctionLibrary {
  private final static String FUN_STRINGTOTABLE = "String to Table";
  // Column naming modes
  enum ColMode {
    AUTO, STRING, STATIC;
  };
  /**
   * Get the list of function descriptors for the functions implemented
   * by this function library. Each command descriptor identifies one
   * function.
   */
  public Vector<IFunctionDescriptor> getFunctionDescriptors() {
    Vector<IFunctionDescriptor> v = new Vector<IFunctionDescriptor> ();
    IFunctionDescriptor fd = FunctionDescriptorFactory.createFunctionDescriptor(
        FUN_STRINGTOTABLE,
        new String[] { "String", "Row Delimiter", "Column Delimiter",
                       "Column Name Mode","Column Names",
                       "Allow Empty Rows/Columns"},
        new String[]
            { "s_arg1", "s_arg2", "s_arg3", "s_arg4", "s_arg5", "s_arg6"},
        IFunctionDescriptor.RETURN_TYPE_TABLE,
        null,
        "This function produces a table from the specified string by using " +
        "the specified row and column delimiters to tokenize the string. If " +
        "the table is to contain only 1 column, do not specify a value for " +
        "Column Delimiter. " +
        "Column names are determined by the \"Column Name Mode\".
         Specify one of: \n" +
        "  - AUTO : Names generated as col0, col1, col2, ...\n" +
        "  - STATIC : Names defined in \"Column Names\", comma seperated\n" +
        "  - STRING : Names defined in first row of \"String\"\n\n" +
        "Use \"Allow Empty Rows/Columns\" to create empty rows/columns for all
        delimiters\n"
        + "  - false (default) : empty tokens will be skipped\n" +
        "  - true : empty tokens will result in empty rows/columns\n");
    v.add(fd);
    return v;
  }
  /**
   * Evaluate a function.
   *
   * @param command Function to evaluate.
   * @param parameters Parameters to function.
   */
  public Object evaluateFunction(String function,  IVariableData parameters) {
    if (function.equals(FUN_STRINGTOTABLE)) {
      return stringToTable(parameters);
    } else {
      return null;
    }
  }
  /**
   * Generate a table from the string parameter passed to function.
   *
   * @param parameters Parameters to function.
   * @return Tabular data
   */
```

```
private ITabularData stringToTable (IVariableData parameters) {
  // See if "Allow Empty Fields" is used.  If not, then return value of
  // stringToTableOld() which preserves old behavior
  // This is needed so as not to break any old StringToTable behavior
  String allowEmptyCells = parameters.getStringVar("s_arg6");
  if (allowEmptyCells == null ||
    allowEmptyCells.equals("") ||
    allowEmptyCells.equalsIgnoreCase("false") ||
    allowEmptyCells.equals("0") ||
    allowEmptyCells.equalsIgnoreCase("no")) {
    return stringToTableOld(parameters);
  }
  // Function arguments
  String inString = parameters.getStringVar("s_arg1");
  String unquoteRowDelim = parameters.getStringVar("s_arg2");
  String unquoteColDelim = parameters.getStringVar("s_arg3");
  String colModeS = parameters.getStringVar("s_arg4");
  String colNames = parameters.getStringVar("s_arg5");
  String colDelim = unquoteColDelim;
  String rowDelim = unquoteRowDelim;
  // Check required values
  if (inString == null || inString.equals(""))
    return null;
  // StringTokenizer will do the right thing
  if ((unquoteColDelim == null) || (unquoteColDelim.length() == 0)) {
    unquoteColDelim = "";
  }
  // The delimiters are treated as a list of chars as delimiters
  // Do this by adding a | between chars
  char[] rowDelimChars = unquoteRowDelim.toCharArray();
  char[] colDelimChars = unquoteColDelim.toCharArray();
  StringBuffer rowDelimSB = new StringBuffer();
  StringBuffer colDelimSB = new StringBuffer();
  // default any regular expression special chars so we can escape them.
  final String metaChars = "^[^\\[\\]\\^\\+\\|\\?\\\\()\\{\\}\\.<>/;\\*%$]*$";
  if (rowDelimChars.length > 0) {
    for (char c : rowDelimChars) {
      // escape any special char
      if (!Pattern.matches(metaChars, String.valueOf(c))) {
        rowDelimSB.append("\\");
      }
      rowDelimSB.append(c);
      rowDelimSB.append("|");
    }
    // remove last '|'
    rowDelimSB.setLength(rowDelimSB.length()-1);
  }
  if (colDelimChars.length > 0) {
    for (char c : colDelimChars) {
      // escape any special char
      if (!Pattern.matches(metaChars, String.valueOf(c))) {
        colDelimSB.append("\\");
      }
      colDelimSB.append(c);
      colDelimSB.append("|");
    }
    // remove last '|'
    colDelimSB.setLength(colDelimSB.length()-1);
  }
  // get the actual, escaped, delimiter regular expressions
  rowDelim = rowDelimSB.toString();
  colDelim = colDelimSB.toString();
  // How are the columns to be named
```

```
ColMode colMode = ColMode.AUTO;
if ((colModeS != null) && (colModeS.length() > 0)) {
  try {
    colMode = ColMode.valueOf(colModeS.trim().toUpperCase());
  } catch (IllegalArgumentException e) {
    // bogus column mode is specified, default to AUTO
    colMode = ColMode.AUTO;
  }
}
// The number of splitted strings in the first row is the number of
// columns in table
int colCount = 0;
String[] rows;
// if no rowDelim, whole string is treated as a row
if (rowDelim.equals("")) {
  rows = new String[] {inString};
} else {
  rows = inString.split(rowDelim, Integer.MAX_VALUE);
}
// if inString is empty, no row is needed
if (inString.equals("")) {
  rows = new String[0];
}
// we do have some rows...
if (rows.length > 0) {
  // if no column delimiter, whole row is one column
  if (colDelim.equals("")) {
    colCount = 1;
  } else {
    // use col delimiter to split it
    colCount = rows[0].split(colDelim, Integer.MAX_VALUE).length;
  }
}
// Initialize table and add columns
ITabularData table = TabularDataFactory.createTabularData();
String[] columnNames = null;
switch (colMode) {
case AUTO:
  for (int i=0; i<colCount; i++) {
    table.addColumn("col" + i,TabularData.COL_TYPE_STRING);
  }
  break;
case STRING:
  // Make sure this is at least one row
  if (rows.length > 0) {
    // 1st row is the colNames
    // we do have some rows...
    // if no column delimiter, whole row is one column, which will be the
    //  col name
    if (colDelim.equals("")) {
      columnNames = new String[] {rows[0]};
      table.addColumn(columnNames[0], TabularData.COL_TYPE_STRING);
     // the 1st row IS
     // the name
    } else {
          // use col delimiter to split it
      columnNames = rows[0].split(colDelim, Integer.MAX_VALUE);
      int n = 0;
      if (columnNames != null) {
        for (String colName : columnNames) {
          table.addColumn(
            (colName.equals("")) ? "col" + n : colName,
               TabularData.COL_TYPE_STRING);
```

```
            n++;
          }
        }
      }
      // since we've used 1st row as column names, remove it from the array
      List<String> rowList = new ArrayList<String>(Arrays.asList(rows));
      rowList = rowList.subList(1, rows.length);
      rows = new String[rows.length-1];
      rows = rowList.toArray(rows);
    }
    break;
  case STATIC:
    // get static column from argument
    columnNames = colNames.split(",", Integer.MAX_VALUE);
    // Figure out the correct number of columns
    int maxCol = 0;
    // if column delimiter is empty, then there is only one column, regardless
    if (unquoteColDelim.equals("")) {
      maxCol = 1;
    } else {
      // If there isn't any row data, just use all columnNames
      maxCol =
        (rows.length > 0 && !rows[0].equals("")) ?
                        colCount : columnNames.length;
    }
    // add column names based on the colNames argument
    int i = 0;
    if (columnNames != null) {
      for (; i < columnNames.length && i < maxCol; i++) {
        String colName = columnNames[i];
        table.addColumn(
          (colName.equals("") ? "col" + i : colName),
                          TabularData.COL_TYPE_STRING);
      }
    }
    // if static col names is shorter, fill up with default column names
    for (; i < maxCol; i++) {
      table.addColumn("col" + i,TabularData.COL_TYPE_STRING);
    }
    colCount = maxCol;
    break;
  }
  // parse string and adding rows to table
  for (int row = 0; row < rows.length; row++) {
    table.addRow("row" + row);
    boolean noColDelimiter = colDelim.equals("");
    if (colCount == 1) {
      table.setCellValue(rows[row], row, 0);
    } else {
      String[] cols;
      // if no col delimiter, whole row is one column, no need to split
      if (noColDelimiter) {
        cols = new String[] {rows[row]};
      } else {
        // do need to split it
        cols = rows[row].split(colDelim, Integer.MAX_VALUE);
        if (cols != null) {
          for (int col = 0; col < colCount && col < cols.length; col++) {
            table.setCellValue(cols[col], row, col);
          }
        }
      }
    }
```

```
    }
    return table;
  }
  /**
   * This is the old StringToTable implementation which uses StringTokenizer,
   * which will by default ignore consecutive delimiters
   * @param parameters
   * @return
   */
  private ITabularData stringToTableOld (IVariableData parameters) {
    // Function arguments
    String inString = parameters.getStringVar("s_arg1");
    String rowDelim = parameters.getStringVar("s_arg2");
    String colDelim = parameters.getStringVar("s_arg3");
    String colModeS = parameters.getStringVar("s_arg4");
    String colNames = parameters.getStringVar("s_arg5");
    // Check required values
    if ((inString == null) || (inString.length() == 0) ||
      (rowDelim == null) || (rowDelim.length() == 0)) {
      return null;
    }
    // StringTokenizer will do the right thing
    if ((colDelim == null) || (colDelim.length() == 0)) {
      colDelim = "";
    }
    // Map special delimiter strings to their internal value
    //      rowDelim = delimValue (rowDelim);
    //      colDelim = delimValue (colDelim);
    // How are the columns to be named
    ColMode colMode = ColMode.AUTO;
    if ((colModeS != null) && (colModeS.length() > 0)) {
      try {
        colMode = ColMode.valueOf(colModeS.trim().toUpperCase());
      } catch (IllegalArgumentException e) {
        // bogus column mode is specified, default to AUTO
        colMode = ColMode.AUTO;
      }
    }
    // The number of tokens in the first row is the number of columns in table
    int colCount = 1;
    StringTokenizer st = new StringTokenizer (inString,rowDelim);
    if ((st.hasMoreTokens())) {
      colCount = new StringTokenizer(st.nextToken(),colDelim).countTokens();
    }
    // Tokenizer for iterating through rows in string
    StringTokenizer rowSt = new StringTokenizer (inString,rowDelim);
    // Initialize table and add columns
    ITabularData table = TabularDataFactory.createTabularData();
    switch (colMode) {
    case AUTO:
      for (int i=0; i<colCount; i++) {
        table.addColumn("col" + i,TabularData.COL_TYPE_STRING);
      }
      break;
    case STRING:
      st = new StringTokenizer (rowSt.nextToken(),colDelim);
      for (int i=0; i<colCount; i++) {
        table.addColumn(st.nextToken(),TabularData.COL_TYPE_STRING);
      }
      break;
    case STATIC:
      st = new StringTokenizer (colNames,",");
      for (int i=0; i<colCount; i++) {
```

```
        if (st.hasMoreTokens()) {
          table.addColumn(st.nextToken(),TabularData.COL_TYPE_STRING);
        } else {
          table.addColumn("col" + i,TabularData.COL_TYPE_STRING);
        }
      }
      break;
    }
    // Parse string adding rows to table
    int row = 0;
    while (rowSt.hasMoreTokens()) {
      table.addRow("row" + row);
      if (colCount == 1) {
        table.setCellValue(rowSt.nextToken(), row, 0);
      } else {
        int col = 0;
        StringTokenizer colSt = new StringTokenizer
         (rowSt.nextToken(), colDelim);
        while (colSt.hasMoreTokens() && (col < colCount)) {
          table.setCellValue(colSt.nextToken(), row, col++);
        }
      }
      row++;
    }
    return table;
  }
}
```

# 5 Defining Dashboard Commands


- Scenario lifecycle ..... 150
- Defining commands ..... 150
- Using dashboard variables in commands ..... 151
- Defining commands for creating a scenario instance ..... 153
- Defining commands for editing a scenario instance ..... 155
- Supporting deletion of a scenario instance ..... 157
- Supporting deletion of all instances of a scenario ..... 159
- Using popup dialogs for commands ..... 160
- Command options ..... 162
- Associating a command with keystrokes ..... 162
- Defining multiple commands ..... 163
- Creating custom commands ..... 164
- Apama set substitution command ..... 167

For users to have full control over their scenario instances, their dashboards need to provide the ability to create, edit, and delete instances of the scenarios. Dashboard Builder allows you to integrate these operations with dashboards.

The sections listed below provide general information about commands, and detail the how to integrate scenario-management commands into a dashboard to create, edit, and delete scenario instances. They also include sections on compound commands and custom commands. The command for sending events to correlators is covered in a separate chapter (see "Defining Send-event Commands" on page 225), as is defining SQL commands (see "Using SQL Data" on page 243).

## Scenario lifecycle

To use a scenario that has been loaded into a correlator, a user must create an instance of it. To create an instance, the user specifies values for all the scenario input variables so that the instance is properly configured. Users can create multiple instances of a scenario, typically with one or more different values for input variables.

When a user creates a scenario instance, he or she is the owner of the instance; by default, other users do not have access to it.

Once created, an instance continues running until complete or deleted by the user (instances can also fail if, for example, a run time error occurs). The values of the input variables can be edited after it has been created to change the characteristics of the instance.

The Create, Edit, and Delete operations are part of the scenario lifecycle. Dashboard Builder enables you to integrate commands with a dashboard so they can be performed by users.

## Defining commands

A command is defined by associating it with an action property of a dashboard object such as a push button. When the action is triggered, in this case, when the button is pressed, the command is performed.

For control objects such as push buttons, commands are defined by setting the `actionCommand` property. For other objects such as labels and charts, the commands are defined by setting the `command` property.

**To see how this works**

1. Create a new dashboard.
2. From the **Controls** tab in the Object Palette, select the **Push Button** object and add it to the dashboard canvas.
3. With the push button object selected, in the Object Properties panel, right-click the `actionCommand` property and select **Define Command > Apama** from the popup menu.

   This displays the Define Apama Command dialog.

4. To define a command, select a command type and specify values for the remaining fields.

   The fields vary based on the command being defined. The common set of fields is as follows.

   - **Command** — The command to be performed when the action is triggered. The command selected will hide or show many of the other fields.
   - **Correlator** — The correlator where the command is to be executed. If creating a new instance of a scenario, this is the correlator where the instance will be created.
   - **Scenario** — The type of scenario being created edited or deleted.
   - **Data Server** — Advanced users can specify the logical name of the Data Server to serve the data for the command execution. See "Working with multiple Data Servers" on page 72 for more information.

   In this documentation, some of the **Define Apama Command** dialogs are shown *without* the **Data Server** field, which was added in a later release.

   The fields in the **Parameters** section are specific to the specified scenario.

   **Note:** When executing commands in display server deployed dashboards, warning and error dialogs are not displayed to warn of error conditions that occur.

## Using dashboard variables in commands

The value of all fields in the Define Apama Command dialog, with the exception of the `Command` field, can be set to dashboard variables. This allows you to dynamically configure the command or set its parameters at run time.

For example, you will typically set the field `Instance` to the dashboard variable `$instanceId`. The `instanceId` field identifies the scenario instance the command is to affect, and the variable `$instanceId` gets set to the unique id of the dashboard's currently-selected scenario instance. If you then trigger a scenario command, the command will affect the instance identified by `$instanceId`, which is the instance selected on the dashboard.

Understanding dashboard variables is essential to being able to add scenario commands to a dashboard. Most commands take parameters that you need to supply values for and in most cases you'll want to prompt the user for the values.

To create an instance of the tutorial, scenario values for the `Instrument` and `Clip Size` variables must be specified. To enable the user to do this, the dashboard needs to include input fields where the values can be specified. These values then need to be used as parameters to the command. This is done through the use of dashboard variables.

To get the value a user has entered in an input field, you need to associate the input field with a dashboard variable so that the variable is updated when the user enters a value in the field. This is done by setting the `varToSet` property of the input field.

**To review the process**

1. Create a new dashboard.
2. From the **Controls** tab in the Object Palette, select the first text field object and add it to the dashboard canvas.
3. From the **Labels** tab in the Object Palette, select the second label object and add it to the dashboard canvas.

   You will now associate the text field with a variable so that when its value changes the label object updates to show the value.
4. Add the dashboard variable `$value` by selecting **Variables** from the **Tools** menu and adding it in the **Variables** panel.
5. Ensure that **Use As Substitution** is checked. Be sure to click the **Add** button to add it the list of variables.

   **Note:** Several substitution variables are automatically created when you create a dashboard.
6. Select the label object and in the Object Properties panel, right click the `valueString` property and select **Attach to Data > Variable**.
7. Select `$value` and click **OK**.

   The label object will contain no text; it is attached to the `$value` variable which has not been set.
8. Select the text field object and in the Object Properties panel, attach its `varToSet` property to the dashboard variable `$value`.
9. Select the `executeOnLostFocusFlag` property and enable it.

   The text field is now bound to `$value`. When text is entered into the field `$value` will change and the label object will update to show the new value. You are now ready to test this.

   Control objects such as text fields and push buttons are not enabled in the Builder canvas. To test these objects, you need to save the dashboard and then select **Tools | Preview Window...**.
10. Type text into the text field object in the preview window, and press **Enter**. The label object will update to show the text that was entered.

Binding control objects to dashboard variables makes the values available for use not only as property attachments but also as parameters to commands. For fields in the **Define Apama Command** dialog, you can either hard code a value by typing it in or select a dashboard substitution variable, such as `$value`, to use as the value. The latter will be

the most common case as control objects such as text fields will typically be used to get the value for command parameters.

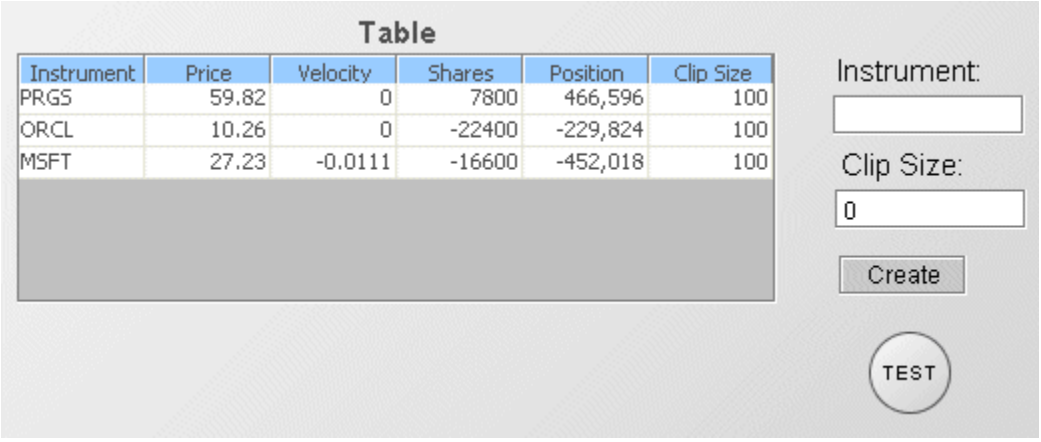
## Defining commands for creating a scenario instance

To add the `Create` function to a dashboard, you need to add control objects such as text fields and check boxes to the dashboard to prompt the user for the values of scenario input variables. You need to then create dashboard variables to hold the values of the control objects and the objects bound to the variables via their `varToSet` property.

You next need to add a control object, such as a push button, to the dashboard to perform the command. In the Define Apama Command dialog, select the command Create scenario instance and use the dashboard variables as the values for the scenario variables.

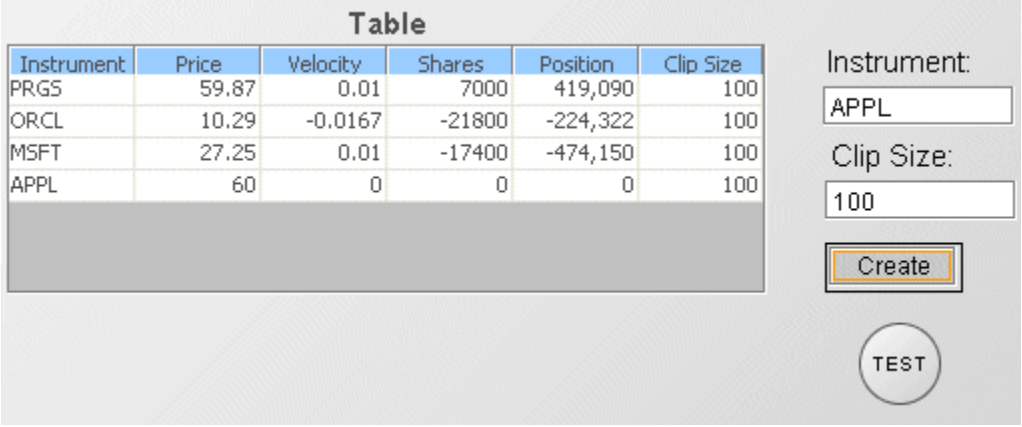
### To define commands for creating a scenario instance

1. Open the file `tutorial-create.rtv` by selecting **Create Instance** in the tutorial main page.
2. Double-click the object labeled **Test** to display the dashboard in a new window such that the control objects are enabled.




This dashboard displays a summary table of all instances of the tutorial scenario and a form for creating new instances.

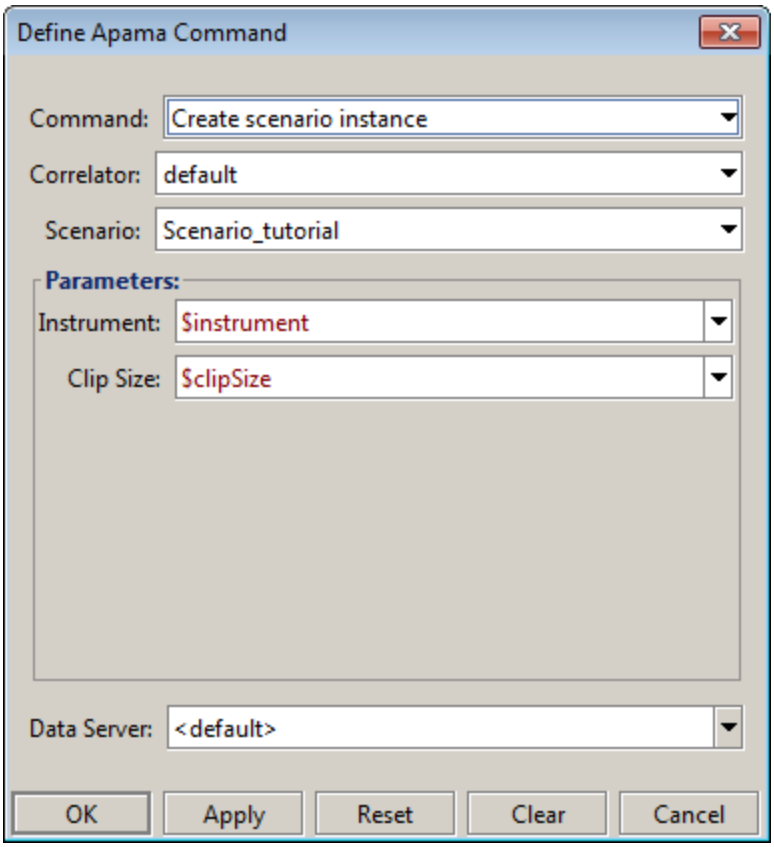
3. In the form enter `APPL` for the `Instrument` and `100` for the `Clip Size` and click the **Create** button. This will create a new instance of the scenario.




This dashboard has the dashboard variables `$instrument` and `$clipSize` defined. The text fields are bound to these such that the variables are set when text is entered in the fields. The `actionCommand` property for the Create button is set to perform the `Create` command and use the value of the variables as command parameters.

4. Select the **Create** button and in the properties panel double-click the `actionCommand` property.

   If the test window is displayed, you need to first close it so that you can select **Create** button in the Dashboard Builder main window.



Here the command is defined to create an instance of the tutorial scenario on the default correlator. You can see that the values for the scenario input variables `Instrument` and `Clip Size` are set to the value of the dashboard variables `$instrument` and `$clipSize`.

When creating a scenario instance you must specify a value for each of the scenario input variables. If you do not, you will receive an error when you try to perform the command.

## Defining commands for editing a scenario instance

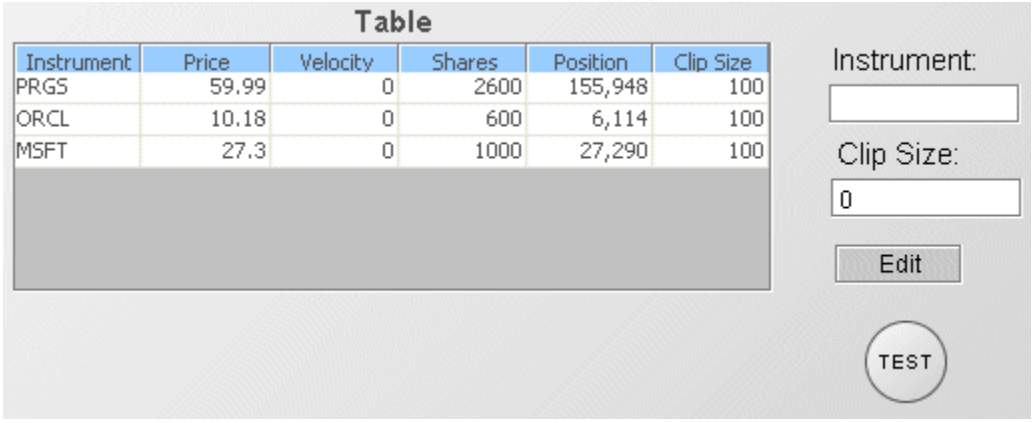
Adding the `Edit` function to a dashboard is similar to you adding the `Create` function. You need to add control objects such as text fields and check boxes to the dashboard to prompt the user for the values of scenario input variables. Then you need to create dashboard variables to hold the values of the control objects and the objects bound to the variables via their `varToSet` property.

You next need to add a control object, such as a push button, to the dashboard to perform the command.

The differences are that when defining the command in the Define Apama Command dialog, you need to identify which instance of the scenario to edit. You also need to identify which scenario variables are to be edited. Unlike the `Create` command, a subset of scenario variables can be changed with the `Edit` command. Users cannot edit scenario variables that have been declared immutable.

### To define commands for editing a scenario instance

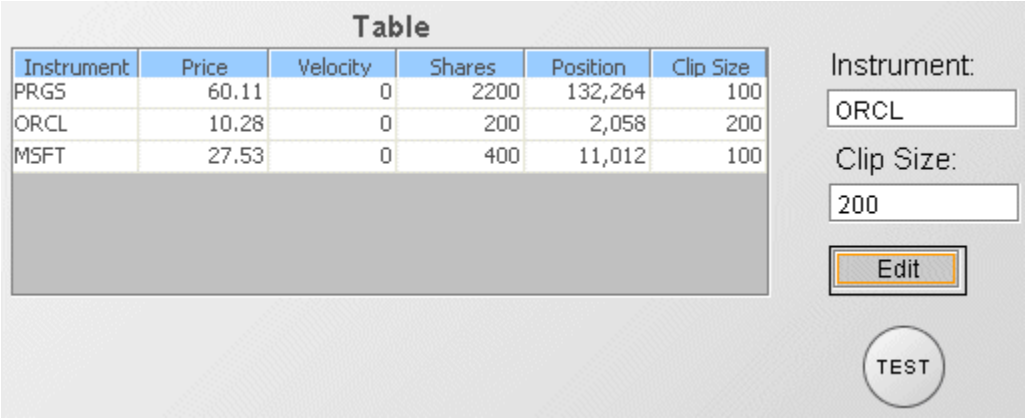
1. Open the file `tutorial-edit.rtv` by selecting **Edit Instance** in the tutorial main page.
2. Double-click the object labeled **Test** to display the dashboard in a new window such that the control objects are enabled..




This dashboard displays a summary table of all instances of the tutorial scenario and a form for editing them.

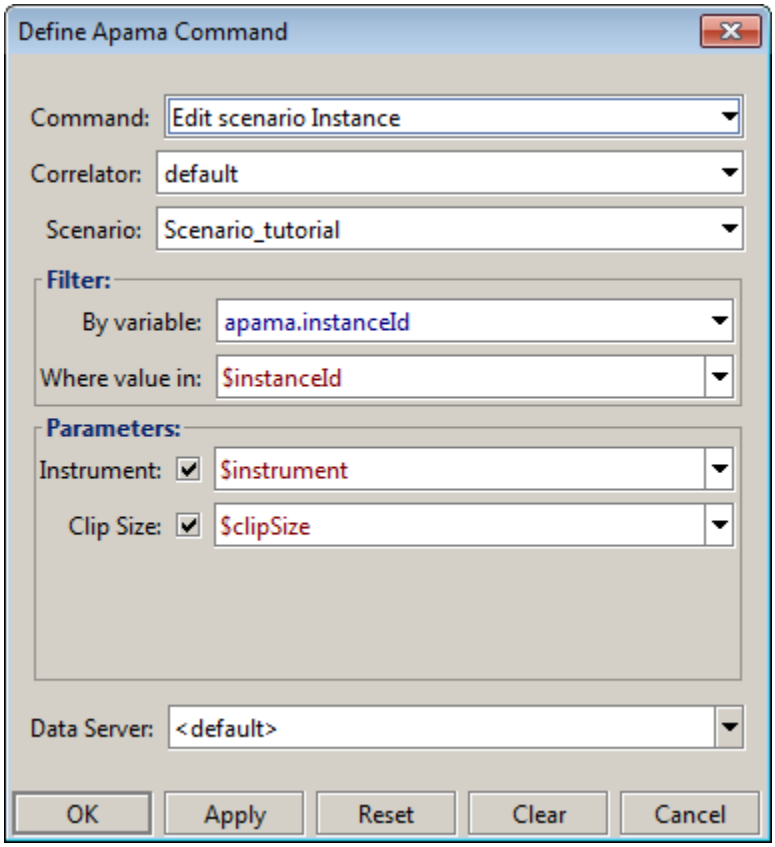
3. Double-click the `APMA` row in the table. This will cause the scenario instance for APMA to become selected and its input variables to be displayed in the form.

4. In the form change the `Clip Size` to `200` and press the **Edit** button. The value of `Clip Size` will change for APMA in the table indicating the scenario instances has been edited.




As in the Create sample, this dashboard has the dashboard variables `$instrument` and `$clipSize` defined. The text fields are bound to these such that the variables are set when text is entered in the fields. The `actionCommand` property for the **Edit** button is set to perform the `Edit` command and use the value of the variables as command parameters.

5. Select the **Edit** button and in the Object Properties panel, double-click the `actionCommand` property.




Here the command is defined to edit the instance of the tutorial scenario whose instance id equals `$instanceId`. You can see that the values for the scenario input variables `Instrument` and `Clip Size` are set to the value of the dashboard variables `$instrument` and `$clipSize`.

The checkbox next to each scenario variable field is used to specify that the variable is to be edited. When performing an `Edit` you do not need to specify values for all scenario variables; only those you want to change.

The `Filter` fields are used to identify the instance to be edited. In this sample `$instanceId` is set when you select a row in the table to the `apama.instanceId` of the selected scenario instance.

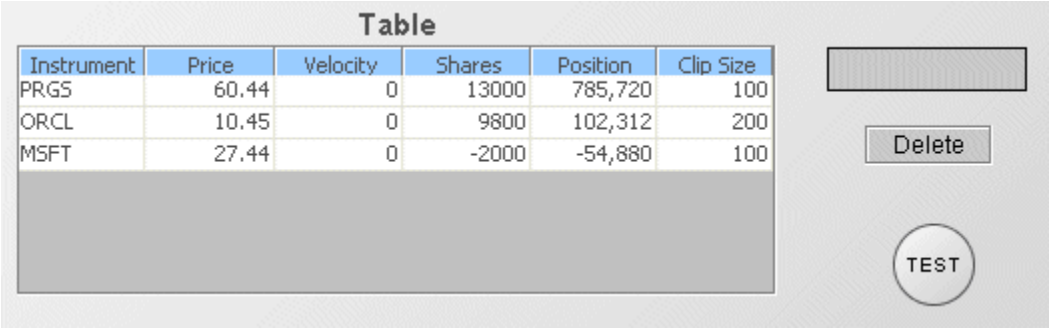
The properties of table and other objects in Dashboard Builder are preconfigured to set `$instanceId` when a drilldown is performed. However, you can use dashboard variables other than `$instanceId` to hold the `apama.instanceId` of a scenario instance.

## Supporting deletion of a scenario instance

**To add the `Delete` function to a dashboard you need to add a control object such as a push button and set its action to perform the delete**

1. Open the file `tutorial-delete.rtv` by selecting **Delete Instance** in the tutorial main page.

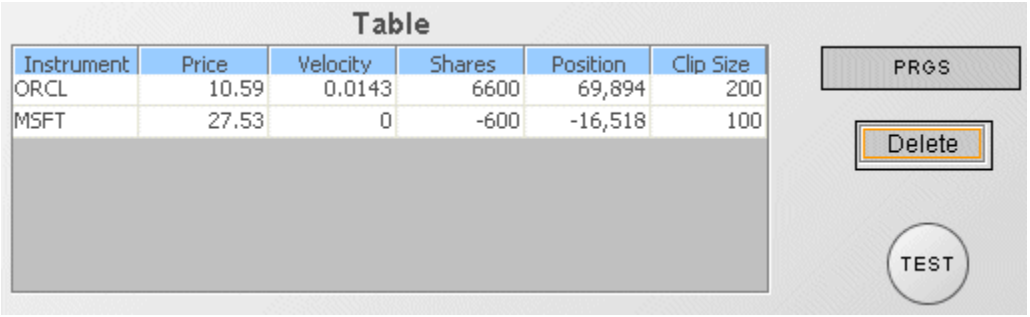
2. Double-click the object labeled `Test` to display the dashboard in a new window such that the control objects are enabled.




This dashboard displays a summary table of all instances of the tutorial scenario and a **Delete** button for deleting the selected instance.

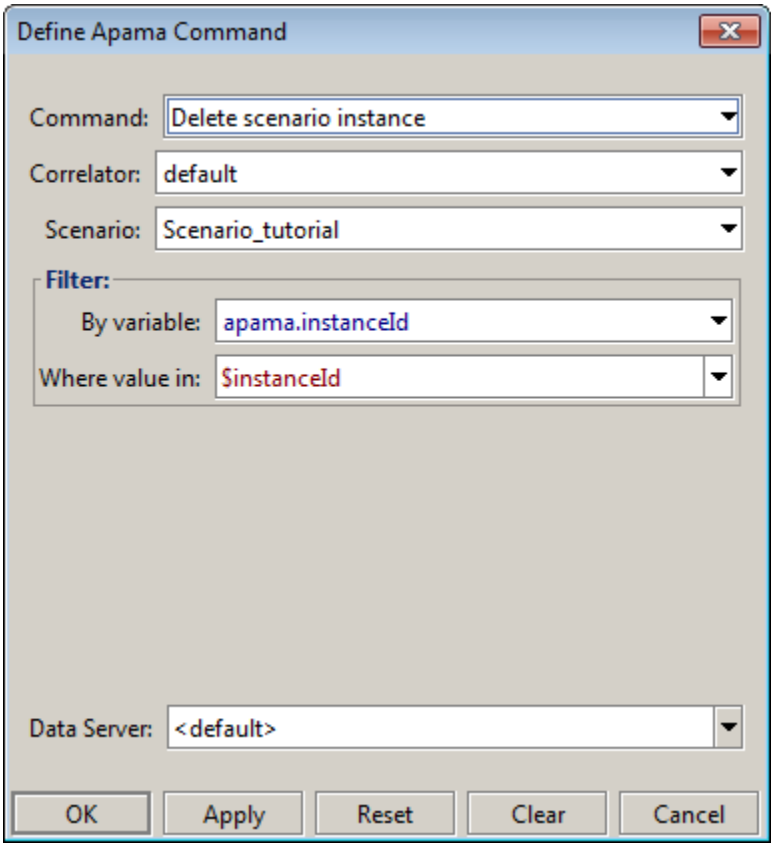
3. Double-click the APMA row in the table. This will cause the scenario instance for APMA to become selected and its Instrument name displayed in the form above the **Delete** button.

4. Click on the **Delete** button. This will delete the APMA instance of the scenario as indicated by the APMA row being removed from the table.




As with `Edit`, when performing a `Delete` you need to identify the instance to be deleted.

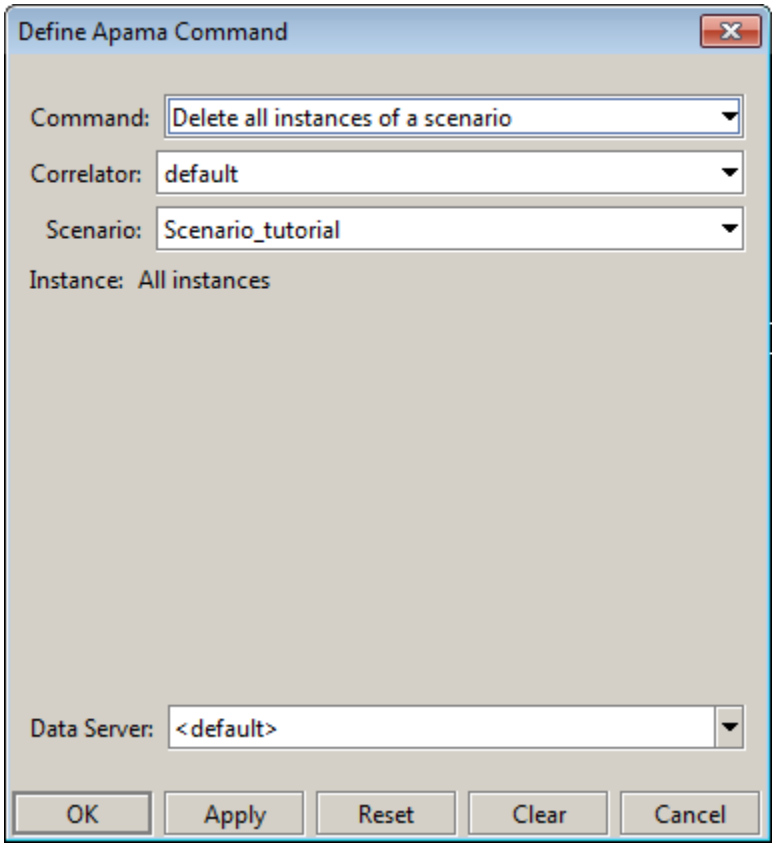
5. Select the **Delete** button and in the Object Properties panel, double-click the `actionCommand` property.




Here the command is defined to delete the instance of the tutorial scenario whose instance id equals `$instanceId`.

## Supporting deletion of all instances of a scenario

For a dashboard you may want to provide an option to delete all instances of a scenario. This can be done by including a control object and setting its action command as follows.



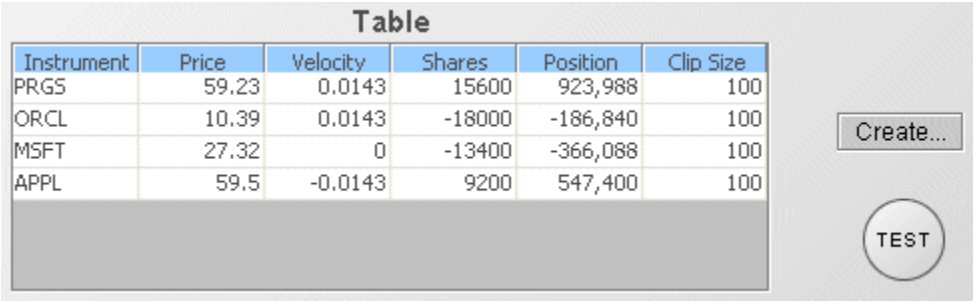

Deleting all instances of a scenario will only delete those instances to which the user has delete access. By default, these are the instances created by the user.

## Using popup dialogs for commands

For the `Create` and `Edit` commands you might not want to integrate the input fields with the main dashboard. They might, for example, occupy space that is better used for display information about running scenarios. An alternative is to place the input fields in separate dialog windows. In this case, the main dashboard contains **Create** and **Edit** buttons. Clicking them displays the appropriate dialogs where users enter the parameters for the command in the input fields and then click the **OK** button to perform the command. You can set up popup dialogs like this in Dashboard Builder.

**To use popup dialogs for commands**

1. Open `tutorial-create-popup.rtv` by selecting **Create Instance Popup** from the tutorial main page.
2. Double-click the **Test** label to display the dashboard in a new window such that the control objects are enabled.




Here the dashboard contains a **Create** button but no fields for setting the input variables.

3. Click on the **Create** button. This will display a dialog window with the fields for creating a scenario instance.

   This dialog is really just another dashboard, in this case `tutorial-create-form.rtv`. The **Create** button displays this dialog by performing a drilldown and displaying `tutorial-create-form.rtv` in a new window.

4. Select the **Create** button object and double-click the `actionCommand` property in the Object Properties panel.

   The command is defined to perform the `Drill Down or Set Substitution` system command. (Note that system commands are not supported for dashboards deployed as applets.)

5. Click on the **Edit Drill Down Target** button.

   The drilldown is set to display `tutorial-create-form.rtv` in a new window.

   The dashboard for the popup dialog was created in Dashboard Builder.

6. Open the file `tutorial-create-form.rtv`.

   You can now select objects in the form and examine their properties in the property panel. The settings are very similar to those in the previous create instance example. The dashboard contains the variables `$instrument` and `$clipSize` which are bound to the text fields. The `actionCommand` property on the `OK` button is defined to perform the create operation using the values of these variables.

   What is different is that when **OK** is pressed, the command will be performed and the dialog window closed. The option to close the window is set in the **closeWindowOnSuccess** property.

7. In the Builder window, select the **OK** button object.

8. Here the **closeWindowOnSuccess** property is enabled. If this property is enabled, the dashboard closes the window that performed the command if the command completes successfully. If the command generates an error, the window will not be closed.

9. The **Cancel** button also has a command associated with it. To see this, select the **Cancel** button object and in the Object Properties panel, double-click the `actionCommand` property.

## Command options

The **Object Properties** pane provides some properties that control some command options:

- **commandCloseWindowOnSuccess** — If enabled, the dashboard will close the window that performed the command if the command completes successfully. If the command generates an error the window will not be closed.
- **commandConfirm** — If enabled, the dashboard will display a confirmation message (specified by the **commandConfirmText** property) before performing the command. It is recommended that this be enabled for delete commands.
- **commandConfirmText** — If **commandConfirm** is enabled, the dashboard will display the value of this property as a confirmation message.

## Associating a command with keystrokes

This chapter's previous examples define commands that are to be invoked by the dashboard users via mouse actions. You can also define commands that are to be invoked by dashboard users via keystrokes.

You do this by adding a HotKey object to the Builder canvas.

**Note:** Thin client, Display Server deployments do not support this feature. With such deployments, users cannot use keystrokes to invoke builder-defined commands. In addition, the HotKey is not supported inside of composite objects.

The HotKey object is located in the **Controls** tab of the object palette:

When you add a HotKey object to the Builder canvas, it does not appear on the end user's dashboard. But as dashboard builder, you set HotKey properties in order to associate keystrokes with a command:

- **hotKey** property: Specify the keystrokes that you want dashboard users to use in order to invoke the command. The value of this property is a text string whose format is described below.
- **command** property: Specify the command to be invoked. Do this as described in this chapter, above.

The hotKey property value must be a text string that consists of a sequence of *keystroke-designators*. A simple keystroke designator is one of the following:

- Function key designator: `F1`, `F2`, `F3`, ..., or `F12`.
- digit or letter: `a`, `b`, `c`, ..., `z`, `0`, `1`, `2`, ..., or `9`.

You can also form a keystroke designator by adding one of the following prefixes to a simple keystroke designator:

- `SHIFT+`
- `CTRL+`
- `ALT+`
- `CTRL+SHIFT+`
- `ALT+SHIFT+`
- `CTRL+ALT+`
- `CTRL+ALT+SHIFT+`

So for example, the keystroke that results from holding down the control and the shift key and striking the F1-function key is designated as follows

```
CTRL+SHIFT+F1
```

And the keystroke that result from holding down the shift key and striking the letter f is designated as follows:

```
SHIFT+f
```

For the dashboard user, when focus is on the dashboard, the specified key sequence triggers execution of the command.

## Defining multiple commands

You can associate multiple commands with an action by using the **Define Multiple Commands** dialog.

**To define multiple commands**

1. Right-click `command` property and select **Define Command > MULTIPLE**.
2. In the **Define Multiple Commands** dialog, choose **APAMA** in the **New Command** combo box, and then click the **Add** button to add an Apama command.

   **Important:** The commands are launched in an arbitrary order, and are executed asynchronously; there is no guarantee that one command will finish before the next one in the sequence starts.

   See "Apama set substitution command" on page 167.

## Creating custom commands

**To provide a Data Server or Display Server with a library of custom commands**

1. Develop an implementation of `com.apama.dashboard.function.ICommandLibrary`. See "Developing a custom-command library" on page 164.
2. Install your implementation. See "Installing a Custom-Command Library" on page 165.

### Developing a custom-command library

A sample implementation of `ICommandLibrary` is included below in "Sample ICommandLibrary implementation" on page 165.

You can find a sample implementation of `ICommandLibrary` in the following file:

```
samples\tutorial\src\com\apama
\dashboard\sample\SampleCommandLibrary.java
```

Your implementation of `ICommandLibrary` must implement the following methods:

- `getCommandDescriptors`: Creates a command descriptor for each function that the library supports; returns a list of `com.apama.dashboard.command.ICommandDescriptors`. This method is called once at Data Server or Display Server startup.
- `invokeCommand`: Performs the command with the specified name, using the specified arguments.

When you compile your implementation, ensure that `ap-dashboard-client.jar` is on your class path. This `jar` file is in the `lib` directory of your Apama installation.

Your implementation of `invokeCommand` can set or retrieve substitution values, if necessary, by using the following methods of `com.apama.dashboard.DashboardManager` and `com.apama.dashboard.IDashboardContext`:

- `DashoardManager.getCommandDashboardContext`: This static method returns an instance of `IDashboardContext`.
- `IDashboardContext.getSubstitution`: Gets the value of a substitution with a given name.
- `IDashboardContext.setSubstitution`: Sets the value of a substitution with a given name.
- `IDashboardContext.setSubstitutions`: Sets the values of substitutions, where the substitutions and values are specified with `String` vectors.

Each set method has a boolean argument, `triggerUpdate`, which controls whether objects attached to the substitution are updated. If it is `false`, they are not. If the

substitutions are only used as command parameters or in drilldowns, you can improve performance by specifying `false`.

Following is an example:

```
import com.apama.dashboard.DashboardManager;
import com.apama.dashboard.IDashboardContext;
...
IDashboardContext ctxt =
   DashboardManager.getCommandDashboardContext();
String val1 = ctxt.getSubstitutionValue("$subst1");
...
ctxt.setSubstitution("$subst2", "val2", false);
```

## Installing a Custom-Command Library

**To install your function library for a given Data Server or Display Server**

1. Include a line in the Data Server or Display Server's `EXTENSIONS.ini` file that specifies the fully qualified name of your `ICommandLibrary` implementation. The line must have the following form:

   ```
   command fully-qualified-classname
   ```

2. Create a `jar` file that contains your `ICommandLibrary` implementation, and either add it to `APAMA_DASHBOARD_CLASSPATH` (changes to this environment variable are picked up by dashboard processes only at process startup) or add it to the list of **External Dependencies** in your project's **Dashboard Properties**. (In Software AG Designer, right-click your project and select **Properties**, expand **Apama**, select **Dashboard Properties**, activate the **External Dependencies** tab, and click the **Add External** button). You can also use the `--dashboardExtraJars` command line argument to specify this `jar` file.

A Data Server or Display Server's `EXTENSIONS.INI` is, by default, located in the `lib` directory of its Apama installation. You can specify a Data Server or Display Server's `EXTENSIONS.ini` file at startup by using the `-X` or `--extensionFile` option — see *Deploying and Managing Apama Applications*.

The `EXTENSIONS.ini` specifies the function library to use. This file identifies all the user supplied extension classes (including function libraries and scenario authorities). Here is a sample `EXTENSIONS.ini`:

```
function com.apama.dashboard.sample.SampleFunctionLibrary
command com.apama.dashboard.sample.SampleCommandLibrary
scenarioAuthority com.apama.dashboard.sample.SampleScenarioAuthority
```

This file installs a function library, a command library, and a scenario authority.

## Sample ICommandLibrary implementation

Below is a sample implementation of `ICommandLibrary`, which you can find under `samples\dashboard_studio\tutorial\src`:

```
package com.apama.dashboard.sample;
import java.util.ArrayList;
import java.util.List;
import javax.swing.BorderFactory;
import javax.swing.JFrame;
```

```
import javax.swing.JLabel;
import com.apama.dashboard.command.CommandDescriptorFactory;
import com.apama.dashboard.command.ICommandDescriptor;
import com.apama.dashboard.command.ICommandLibrary;
/**
 * SampleCommandLibrary is an example of a custom command library for
 * Dashboard Builder. Custom commands allow you to extend Dashboard Builder
 * to run custom code in response to a user action such as a clicking on
 * a button.
 * <p>
 * SampleCommandLibrary implements the commands:
 * <ul>
 * <li><b>Show Message</b>: Displays a message window showing the arguments
 * passed to the command.
 * </ul>
 *
 * $Copyright(c) 2013 Software AG, Darmstadt, Germany and/or its licensors$
 *
 * @version      $Id: SampleCommandLibrary.java 84623 2008-06-25 22:41:10Z cr $
 */
public class SampleCommandLibrary implements ICommandLibrary {
private final static String CMD_ECHO = "Show Message";
  /**
   * Get the list of command descriptors for the commands implemented
   * by this command library. Each command descriptor identifies one
   * command.
   */
  public List<ICommandDescriptor> getCommandDescriptors() {
    List<ICommandDescriptor>  v = new ArrayList<ICommandDescriptor> ();
    v.add(CommandDescriptorFactory.createCommandDescriptor(CMD_ECHO));
    // Add additional command descriptors here.
    return v;
  }
  /**
   * Execute a command.
   *
   * @param command Command to execute.
   * @param parameters Parameters to command.
   */
  public boolean invokeCommand(String command, Object parameters) {
    if (command.equals(CMD_ECHO)) {
          //Create and set up the window.
          JFrame frame = new JFrame("Message");
          frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
          //Add the ubiquitous "Hello World" label.
          JLabel label = new JLabel(parameters.toString());
          label.setBorder(BorderFactory.createEmptyBorder(30,100,30,100));
          frame.getContentPane().add(label);
          frame.setLocation(100,100);
          //Display the window.
          frame.pack();
          frame.setVisible(true);
    } else {
      // Add additional command handlers here.
    }
    return true;
  }
}
```

## Apama set substitution command

Use the Apama set substitution command to set substitution values without using the **Drill Down or Set Substitution** system command.

**To set substitution values**

1. Right-click the command property and select **System**.
2. In the **Command Type** combo box of the **Define System Command** dialog, select **Execute Custom Command**.
3. In the **Command Name:** field, type `Apama_SetSub1.0`.
4. In the **Command Value:** field, type a string in the following format:

   ```
   Sub=Value[;Sub=Value...]
   ```

   For example, to set `$MySub1` to `value1` and `$MySub2` to `value2`, enter the following command value:

   ```
   MySub1=value1;MySub2=value2
   ```

   Remember to remove the `$` from the substitution name.

# 6 Reusing Dashboard Components

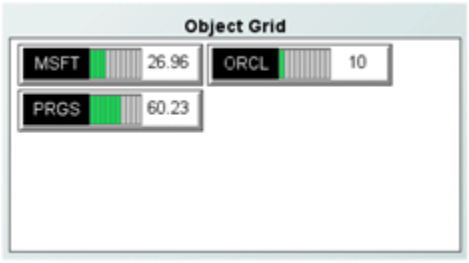

- Using Object Grids ..... 170
- Using Composite objects ..... 174
- Using Composite Grids ..... 181
- Using include files ..... 186
- Working with multiple display panels ..... 188

As the number and complexity of your dashboards grow, you need the ability to modularize dashboard components into manageable and reusable sets. This allows you to efficiently develop and maintain your dashboards.

This chapter describes the features of Dashboard Builder that allow you to create reusable dashboard components and expand beyond the Table object for the rich display of tabular data.

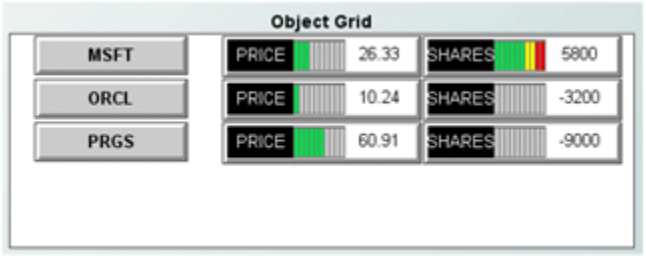
## Using Object Grids

The Object Grid allows you to display tabular data using one or more other object types to show the values of scenario variables or DataView fields. An Object Grid is, as the name implies, a grid of objects. Following is an example of an object grid from the Object Grid tutorial sample:




Here the grid is using one of the label object types in order to display the **Instrument** and **Price** variables of the tutorial scenario instances. The label object used provides a graphical indication of the price as well. There is one instance of the label object for each instance of the scenario. If a new instance of the scenario were created an entry for it would automatically be added to the object grid.

Most objects that appear in the object palette can be displayed in the object grid. Exceptions include tables, some graphs and some general objects. More than one object can be used to visualize each row in the tabular data.



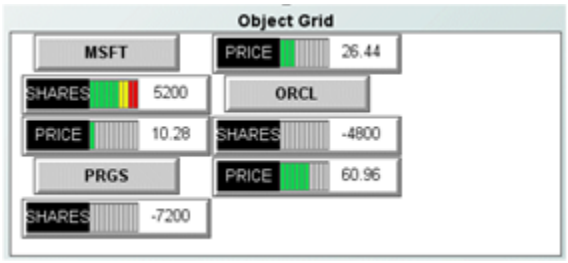

The grid above uses three objects to display the **Instrument**, **Price**, and **Shares** variables of the scenario instances.

Object grids provide one alternative to table objects for visualizing tabular data. They are simple to use but provide limited control over the layout of the objects:

- Objects within a grid are each given the same space as the largest object in the grid.

- Objects within a grid are positioned using a flow layout; positioning objects in the top-left corner of the grid and progressing to the right and bottom.

The following illustrates the layout behavior of the object grid:

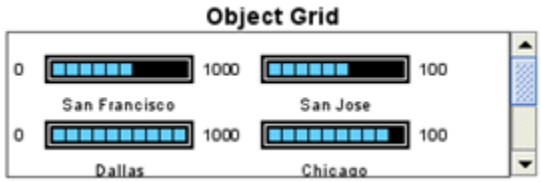



This is the same object grid as in the previous illustration. The only change is that it was resized to be slightly narrower which caused the flow layout of objects to change.

If precise control over the layout of objects is required use the Composite or Composite Grid objects.

## Configuring Object Grids

The Object Grid is in the **Composite** tab of the object pallet.



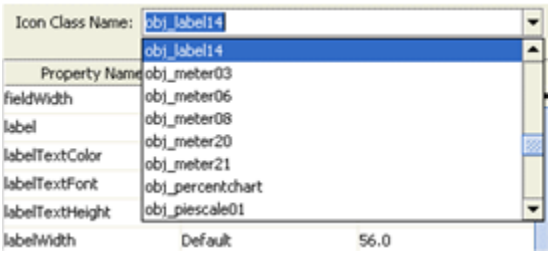

The Object Grid is initialized to display the same sample data as the Table object. The sample data contains seven rows so there are seven instances of the object in the grid.

After adding an Object Grid to your dashboard, you need to attach its `valueTable` property to the tabular data that you want to display. It can be attached to any tabular data source, including the following:

- Apama scenario instance tables
- Apama DataViews
- Dashboard functions that produce tabular data
- Tabular XML data

The `iconProperties` property is used to select and configure the objects that are displayed in the grid. With the grid object selected, in the Object Properties panel, double-click the `iconProperties` property to display the Icon Properties dialog.

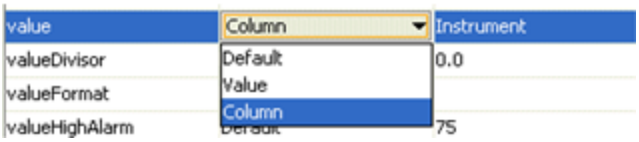
By default the Object Grid is configured to display a single object for each row in its tabular data. The **Icon Class Name** field is where you select the type of object that you want to display in the grid:



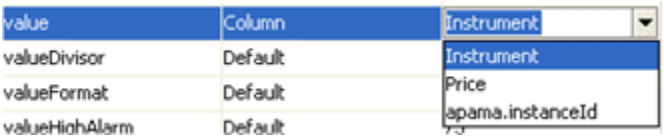

The properties listed correspond to the properties of the object type selected. Properties in the **Icon Properties** dialog can have their value set in one of three ways. How a property is being set is indicated in the **Map** column of the property list:

- **Default**: The property will take the default value. If the default value changes in a future version of Dashboard Builder, the property will take the new default.
- **Value**: The property has a user-supplied value. This value will be the same for all instances of the object displayed in the grid.
- **Column**: The property value comes from the tabular data that the grid object is attached to. Each instance of the object in the grid corresponds to one row in the tabular data. Binding a property to a column causes each instance of the object to use the value of that column in the corresponding row in the tabular data.

If you click in the **Map** column for a property, the Builder displays a list that you can use to select how the property value is set.




If you select **Value**, the value for the property is entered in the **Property Value** column. If you select **Column**, clicking in the **Property Value** column will display a list of all the columns in the tabular data.




When you bind a property to a column in the tabular data, each instance of the object displayed in the grid has that property bound to the value of that column in the corresponding row in the tabular data.

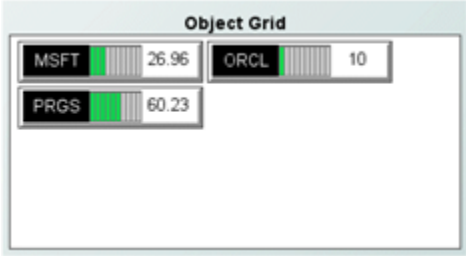
To display multiple objects for each data row, enable the **Allow multiple icon types** check box at the bottom of the Icon Properties dialog.


When enabled, the dialog will change to allow you to add multiple objects for display in the grid.

Use **Add Icon** and **Delete Icon** buttons to add and remove objects from the grid.

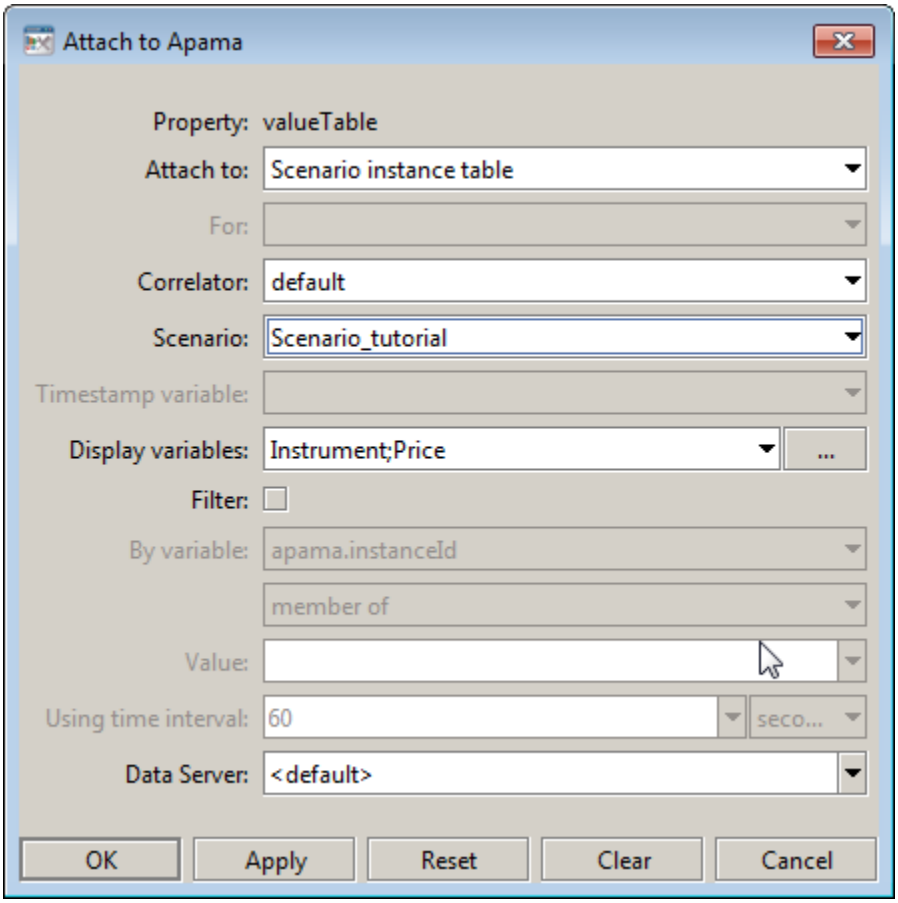
## Recreating the Object Grid sample

The Dashboard Builder tutorial includes an example of the Object Grid, which you can view by double-clicking **Object Grid** on the tutorial main page. This displays the file `tutorial-object-grid.rtv`.

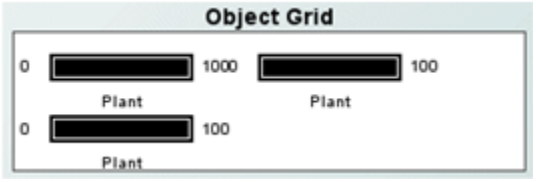



**To recreate this sample, create a new dashboard and perform the following steps**

1. Add an Object Grid to the dashboard and attach its `valueTable` property to the tutorial scenario as follows.

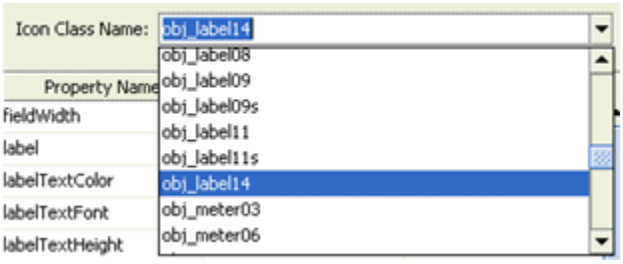



The grid object will update and display as follows:

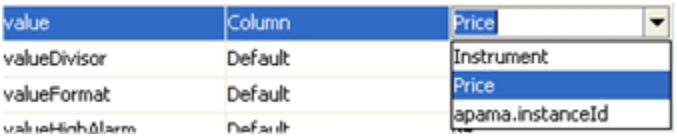



Unless you have separately created or deleted instances of the tutorial scenario, there will be three instances of the scenario and the grid will display three instances of the object. The objects do not show any values from the tutorial scenario because none of their properties have been bound to it in the Icon Properties dialog.

2. Select the grid object and double-click the `iconProperties` property to display the Icon Properties dialog. In the dialog select **obj_label14** as the **Icon Class Name**.

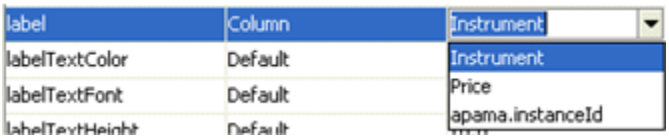



3. In the **Icon Properties** dialog click in the **Map** column of the `value` property and select the type **Column**. Click in the **Property Value** column and select **Price**.




This sets the `value` property of each instance of the object to the value of the **Price** variable in the corresponding instance of the tutorial scenario.

4. Similarly for the `label` property, set the **Map** column to **Column** and select **Instrument** as the value.



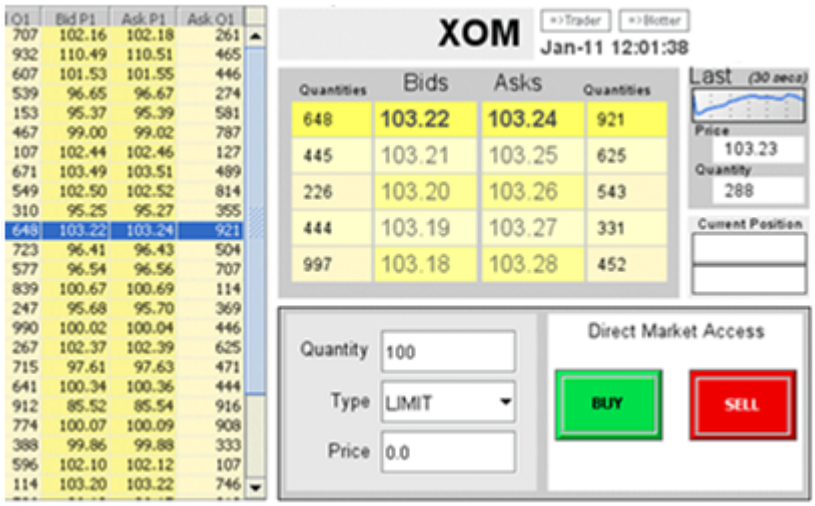

The dashboard should now appear similar to the Object Grid tutorial.

## Using Composite objects

The Composite object allows you to display an `rtv` file as an object within another `rtv` file. This is a powerful capability which allows a complex dashboard to be subdivided into multiple components that can be independently developed and reused in multiple dashboards. The following illustration shows an example of a bid and ask depth display:

| Quantities | Bids | Asks | Quantities |
|---|---|---|---|
| 648 | 103.22 | 103.24 | 921 |
| 445 | 103.21 | 103.25 | 625 |
| 226 | 103.20 | 103.26 | 543 |
| 444 | 103.19 | 103.27 | 331 |
| 997 | 103.18 | 103.28 | 452 |

This can be created and saved in an `rtv` file and with the Composite object be used in one or more other dashboards.




Here the bid and ask display is shown in a Composite object combined with other objects to form a complete dashboard.

**Note:** The HotKey (see "Associating a command with keystrokes" on page 162) is not supported inside of composite objects.

## Creating files to display in composite objects

While the Composite object can display any `rtv` file, reusable `rtv` files are typically parameterized. When you select the `rtv` for display in a Composite object, the Composite object will expose as properties all the variables defined in the `rtv` file. These variables are the parameters to the file, and can be set as needed for each use of the file.

As a simple illustration consider an `rtv` file with a single label object, where you want the text of the label, its color, and its font to be configurable whenever the file is used in a Composite object.

To do this, define variables in the `rtv` file for each of these properties, such as the following:

- **labelText**
- **labelColor**
- **labelFont**

In the properties panel, attach the `label`, `labelTextColor`, and `labelTextFont` properties of the label object to these variables.

| | |
|---|---|
| label | local labelText |
| labelTextColor | local labelColor |
| labelTextFont | local labelFont |

When you use these variables in a Composite object, you'll be able to set values for each in order to configure the appearance of the label.

When you edit properties of a Composite object, the property panel attempts to display the appropriate editor based on the name of the variable. Therefore, when you name variables for fonts and colors, end them with **Font** or **Color**, for example **labelColor**.

Variables that will be used for tabular data must have the data type **Table**.

When you define a variable, if you do not want to expose it as a property in a Composite object, uncheck the **Public** attribute of the variable in the **Variables** panel.


**Note:** Substitutions defined in an `rtv` file are not exposed as properties when the file is used in a Composite object. Variables that you want exposed as properties cannot be defined as substitutions, hence they can't start with a **$**.

**Note:** Variable names cannot conflict with the names of properties of the Composite object; variables whose names conflict with Composite-object property names will not be exposed as properties. For example, you cannot have a variable **label** in a file displayed in a composite. The name conflicts with the `label` property of the Composite object.

## Configuring Composite objects

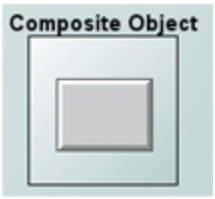
The Composite object is in the **Composite** tab of the object pallet.


The Composite object is initialized to display one row of the same sample data as the Table object. The `rtv` file displayed contains three objects to show the city and unit statistics.

After adding a Composite object to a dashboard you need to specify the `rtv` file to display. The `rtvName` property is used to select the file.

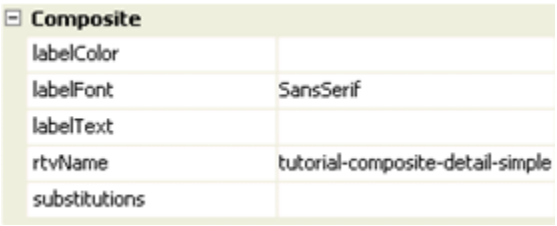

**Note:** This `rtv` file must not itself contain any composite objects. You cannot nest composite objects.

When you select a file, the Composite object is redrawn in order to display the contents of the file.




Note that the Composite object is resized to the size specified in the file being displayed. When you create the file, set its Background Properties to the desired size and color.

The property panel for the Composite object will update to show as properties all the variables defined in the selected file.




Here the **labelColor**, **labelFont**, and **labelText** variables are exposed as properties. Setting these you can change the appearance of the Composite object:

Here the `labelColor` is set to `red`, the `labelFont` to `SanSerif Bold` and the `labelText` to the string `Test`.

Composite object properties that expose variables do not need to be set to static values. You can attach them to any data source, including a scenario or DataView. When so attached, a property's corresponding variable changes whenever the attached data changes. Properties in the `rtv` file that are attached to the exposed variables will update in turn. The following dashboard illustrates this:

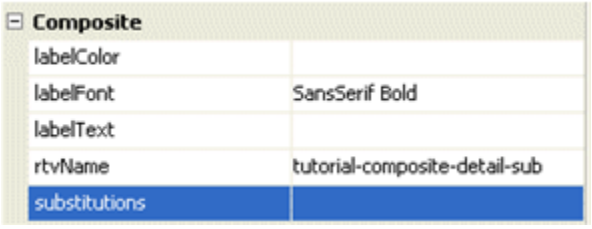

This dashboard consists of a table object showing all instances of the Tutorial scenario and a Composite object containing a single label. This Composite object shows the current price of whatever instrument is selected in the table. Whenever the price changes the composite object updates to show the current price.

As in previous examples, the `rtv` file that is displayed in the composite has the variable **labelText**, which is exposed as a property on the Composite object. The label in the file is attached to **labelText** such that it will show its value. Rather than supplying a static value for the corresponding property `labelText` in the Composite object, this dashboard has `labelText` attached to the selected instance of the Tutorial scenario.

By attaching the property to the scenario and filtering by $instanceId, the `labelText` property will update whenever the value of the attachment changes. When the `labelText` property changes, it will change the value of the corresponding variable in the `rtv` file displayed in the composite which will in turn be reflected in the label.

## Using substitutions with Composite objects

The Composite object supports the setting of substitutions on the file displayed in the composite. It has a single `substitutions` property where the name and value of one or more substitutions can be specified.




Substitutions are specified as a string with the following syntax:

```
$subname:subvalue $subname2:subvalue2 …
```

- If a substitution value contains a single quote character, it must be escaped using a backslash.

```
$subname:/'Quoted Value/'
```

- If a substitution value contains a space, the entire value must be enclosed in single quotes. Do not escape these single quotes.

```
$subname:'Value with Spaces'
```

- The substitution names should not contain any of the following characters:

  Colon (:), pipe (|), period (.), comma (,), semi-colon (;), equals (=), brackets (< >, ( ), { }, [ ]), quotation marks (' "), ampersand (&), slashes (/ \)

**Note:** Substitutions and variables in a Composite object are scoped to the object. If a dashboard contains a Composite object, and both the dashboard and the Composite object have the substitution **$mySub** defined, changes to the value of one will not affect the other. The Composite object will have its own value as will the dashboard.

When you use a Composite object to display detailed information on a selected scenario instance or DataView item, it is often easiest to set the substitution **$instanceId** on the composite. Setting **$instanceId** allows you to define in the `rtv` file displayed in the composite attachments and commands which filter on **$instanceId** as you normally would in other dashboards.

For this use case, the simplest way to set **$instanceId** as a substitution on a composite is to attach the `substitutions` property of the composite object to the **apama.substitutions** variable of the selected instance.

The value of **apama.substitutions** is a string formatted for use as the value of the `substitutions` property. An example of the value for an instance of the tutorial scenario is the following:

```
$instanceId:default.tutorial.21
```

This results in the substitution **$instanceId** being set to `default.tutorial.21` in the `rtv` file displayed in the composite. Attachments and commands filtering on **$instanceId** would be tied to this instance.

**Note:** If the file displayed in a Composite object has buttons or other objects which execute Apama commands to edit or delete an instance of a scenario, you need a substitution set to the ID of the instance. This substitution can be used as the filter on the command to identify the instance that the command operates on. The standard substitution to use is **$instanceId**.

**Note:** Variables cannot be used in filters on attachments or commands. You cannot define a variable **instanceId**, use it in the filter, and set the value as a property on a composite object.

## Composite object interactivity

The Composite object has the `command` and `drillDownTarget` properties. These operate as with other objects, allowing you to define commands and drilldowns that are executed when the object is clicked on.

If the file displayed in the composite contains objects with their `command` or `drillDownTarget` properties set, these will take precedence over those defined on the Composite object.

The following illustration is of a Composite object displaying an `rtv` file with one label object:


The label object **Command A** is defined to run some command, *command A*, and the `command` property of the composite is set to run a different command, *command B*. Clicking on the label object will run *command A*, since the command property in the `rtv` file overrides that of the composite. Clicking on the dark blue background of the composite will run *command B*.

## Composite object sample

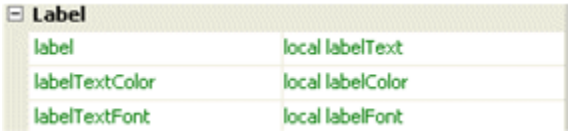
The Dashboard Builder tutorial includes an example of the Composite object.

- Open the file `tutorial-composite-simple.rtv` by selecting **Composite Simple** on the tutorial main page.

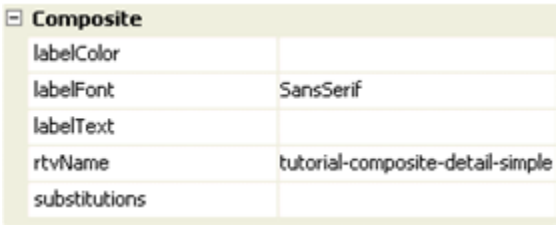

This dashboard displays a composite object with its `rtvName` property set to the file `tutorial-composite-detail-simple`.

- Open the file `tutorial-composite-detail-simple.rtv` in the dashboard Builder.

Examine the label, labelTextColor, and `labelTextFont` properties to see that they are attached to variables.




From the **Tools** menu select **Variables** to see that the variables **labelText**, **labelColor**, and **labelFont** are defined as public variables. These variables are set as properties on the Composite object.




***Recreating the Composite object sample***

**To recreate this sample, create a new dashboard and perform the following steps**

1. Add a Composite object to the dashboard and set its `rtvName` property to `tutorial-composite-detail-simple`. The list of properties in the property panel for the Composite object will update.
2. Set the `labelColor` to `red`, the `labelFont` to `Sans Serif Bold`, and the `labelText` to `Test`.

   The dashboard should now appear similar to the Composite Simple tutorial. Experiment with adding new variables to `tutorial-composite-detail-simple.rtv` and attach object properties to these. In the Composite object, experiment with attaching properties and substitutions to the tutorial scenario.

## Using Composite Grids

The Composite Grid object combines the capabilities of the Composite and Object Grid objects to provide a powerful and flexible means to display multiple scenario instances or DataView items.

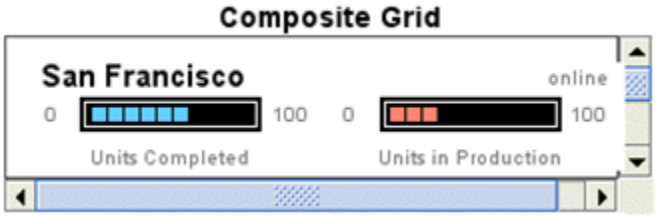

Above, a Composite Grid is used to display the instances of the tutorial scenario. The `rtv` file displayed in the grid contains a set of objects to display the details of a single instance of the tutorial scenario.


The objects are attached to the tutorial scenario filtering on **$instanceId** to select a single instance. The Composite Grid object is configured to pass each instance a unique value of **$instanceId** such that there is one row in the grid for each instance of the scenario.

**Note:** The Composite Grid object is really just an Object Grid with the **Icon Class Name** in its `iconProperties` set to **obj_composite**. The Composite Grid has all the behaviors of the Object Grid and Composite objects.

## Configuring Composite Grids

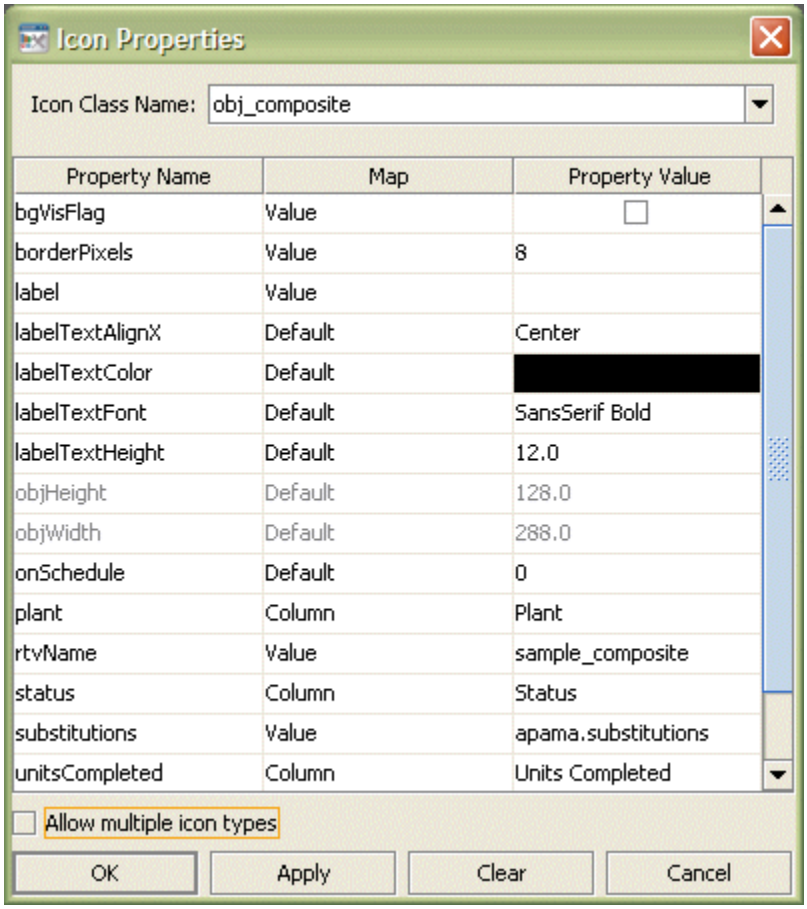
The Composite Grid is in the **Composite** tab of the object pallet.




The Composite Grid is initialized to display the same sample data as the Table object. The sample data contains seven rows so there are seven instances of the object in the grid. For each row of data the Composite Grid displays an `rtv` file containing several objects to show the city and unit statistics.

After adding an Object Grid to your dashboard you need to attach its `valueTable` property to the tabular data to display. See the Object Grid section for details.

The `iconProperties` property is used to configure the Composite object displayed in the grid. With the grid object selected, in the Object Properties panel, double-click the `iconProperties` property to display the Icon Properties dialog.




Notice that **Icon Class Name** is set to **obj_composite**.

Within the Icon Properties dialog set the `rtvName` property to the name of the `rtv` file to display in the Composite. The list of properties will update to show as properties all the public variables in the selected `rtv` file.

**Note:** If the list of properties does not update, close the Icon Properties dialog and redisplay it.

The properties of the Composite object can now be configured in the Icon Properties dialog as needed.

The `substitutions` property is preset to the value of **apama.substitutions**.


The effect of this is to set the substitution **$instanceId** uniquely for each instance of the Composite object displayed in the grid. Each instance will have **$instanceId** set to a unique instance of the scenario or DataView that the Composite Grid is attached to.

## Composite Grid sample

The Dashboard Builder tutorial includes an example of the Composite Grid object.

- Open the file `tutorial-composite-grid.rtv` by selecting **Composite Grid** on the tutorial main page.


This dashboard displays in a grid a composite object with its `rtvName` property set to the file `tutorial-composite-grid-detail.rtv`.

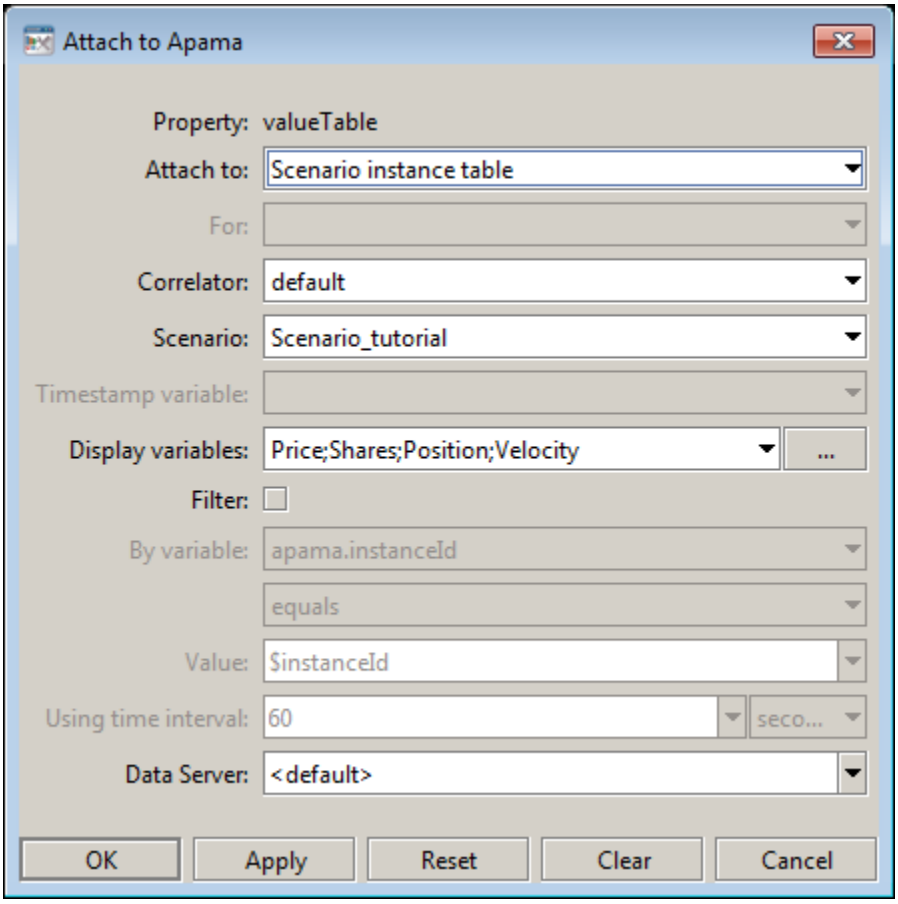
- Open the file `tutorial-composite-grid-detail.rtv` in the Dashboard Builder.


Examine the `label`, and `value` properties of the objects to see that they are attached to the tutorial scenario filtering on **$instanceId**. Range Dynamic objects are used to show **Shares**, **Position**, and **Velocity**. These objects are configured to change color to show if the value is positive, negative, or zero.

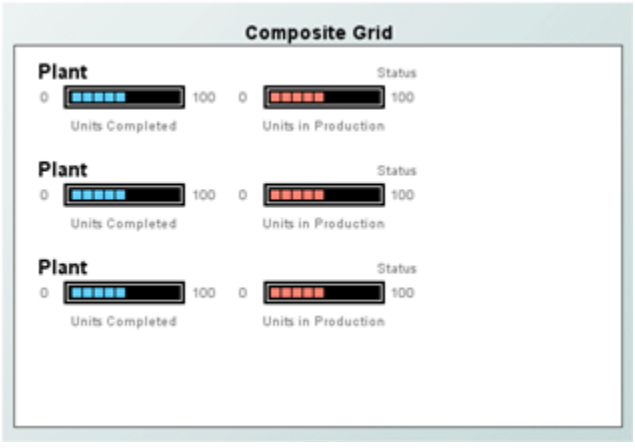
### *Recreating the Composite Grid sample*

**To recreate this sample create a new dashboard and perform the following steps**

1. Add a Composite Grid object to the dashboard.
2. With the Composite Grid selected, in the Object Properties panel select the `valueTable` property and attach it to the tutorial scenario as follows:




The Composite Grid will be similar to the following:




Unless you have created or deleted instances of the tutorial scenario, there will be three instances of the Composite object in the grid. They do not show any data because the sample `rtv` file is not attached to the data of the tutorial scenario.

3. With the Composite Grid selected, double-click the `iconProperties` property in the Object Properties panel. This will display the Icon Properties dialog.

In the Icon Properties dialog set the `rtvName` property to `tutorial-composite-grid-detail`. Close the Icon Properties dialog.

The dashboard should now appear similar to the Composite Grid tutorial.

## Using include files

The Dashboard Builder include file feature provides a way to partition dashboard development and to reuse content in multiple dashboards. It allows you to include in a dashboard the objects, functions, and variables of another `rtv` file. Unlike the Composite object, the included file is not in an object; rather, the contents of the included file are added to the dashboard.

A common use of include files is for navigation controls and status bars that are part of multiple dashboards. These objects could be defined in the `rtv` file `statusbar.rtv`. The file `statusbar.rtv` could then be included in another dashboard.

To include an `rtv` file in a dashboard, select **Include Files** in the **Tools** menu. This will display the Include Files dialog.

The **Add** and **Remove** buttons are used to add and remove included `rtv` files. More than one `rtv` file can be included, and the included files can themselves include other files. However, a file will only be included once.

All the objects, functions, and variables that are defined in an included file become part of the dashboard. Within the Dashboard Builder these are, with one exception, read-only. They appear, can be copied, and can be used in attachments, but they cannot be modified. To modify included elements, open the file containing them in the Dashboard Builder.

The exception is for the initial value of an included variable. Within a dashboard, you can override the initial value of included variables. Consider, for example, an included file that contains a label that is attached to the variable **headerTitle**. When you include this file in a dashboard, the value of the variable **headerTitle** can be set to the title of the dashboard.

*Note:* If objects from an included display file have the same value for the `objName` property as objects in the current display, or other included displays, this may cause links to attach to the wrong object. To avoid this overlap, assign a unique value to the `objName` property of objects in files that you intend to include in other displays.

The background properties such as **Model Width**, **Model Height**, and **bgColor** of included files are ignored.

Substitutions such as **$instanceId** may be used in attachments in included files. Substitutions and variables in included files are scoped to the including dashboard. Runtime changes to their values will be reflected in included objects and functions. An attachment in an included file filtering on **$instanceId** will update whenever **$instanceId** changes in the dashboard.

## Include File sample

The Dashboard Builder tutorial includes an example of Include Files.

- Open the file `tutorial-include-sub.rtv` by selecting **Include File Subs** on the tutorial main page.

  This dashboard uses a status bar defined in an included file. The status bar contains an indicator of correlator connectivity and objects to show the instrument, price, and other variables of the selected instance of the tutorial scenario.

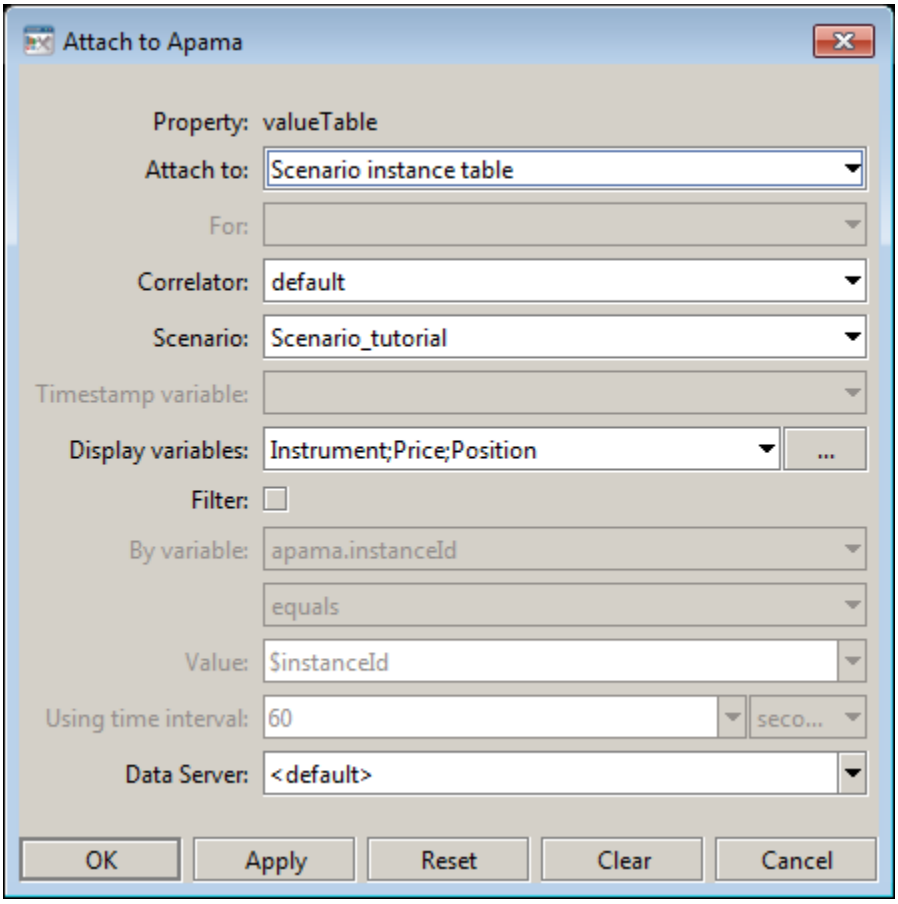
- Open the file `tutorial-include-sub-background.rtv` in the Dashboard Builder.

  Examine the attachments of the `value` and `label` properties of the objects. Notice that they are attached to the tutorial scenario and filtering on **$instanceId**. No values are displayed because **$instanceId** does not have a value. It is set in the dashboard that includes this file.

### *Recreating the Include File sample*

**To recreate the Include File sample create a new dashboard and perform the following steps**

1. Add a Table object to the dashboard and attach its `valueTable` property to the tutorial scenario as follows:




2. Select **Include Files** from the **Tools** menu and add `tutorial-include-sub.background.rtv`.

The dashboard should now appear similar to the Include File tutorial. Double-click a row in the table to see values displayed in the included status bar.

## Working with multiple display panels

It is possible to deploy several displays arranged in separate panels in a single window. Multiple panels are useful when you want to view multiple displays from a top level entry point or if you need to include a navigation panel. To define a window with multiple display panels, create an XML file, a *panels-configuration file*, that specifies a panel layout for a deployed dashboard.

You can supply the location of the panels-configuration file to the Deployment Configuration Editor (see "Using the Deployment Configuration editor" on page 643) or to the Dashboard Viewer executable by using the `-C` or `--panelConfig` option (see "Startup Options for the Dashboard Viewer" on page 709).

The name of a panels-configuration file must have the `.ini` extension. By default, the Display Viewer looks for the `PANELS.ini` file in the current directory. If a panels-

configuration file is not found in the current directory, the Display Server and Display Viewer look for it in the `lib` directory of your Apama installation directory.

## About the format of the panels-configuration file

The panels-configuration file is in XML, and must start with the following:

```
<?xml version="1.0" ?>
<panels xmlns="www.sl.com" version="1.0">
```

The panels-configuration file must end with the following:

```
</panels>
```

In this release, a new set of tags are allowed in the panels-configuration file. These tags are different from the tags that were allowed in previous releases. Previously allowed tags are still allowed. However, new tags and old tags cannot be in the same panels-configuration file.

For information about the new tags, see "Using new tags to configure the panels in a window" on page 189.

For information about the old tags, see "Using old tags to configure the panels in a window" on page 213.

## Using new tags to configure the panels in a window

When using the new tags each panels-configuration file must contain exactly one `rtvLayout` tag. The `rtvLayout` tag encloses the tags that define the multiple displays. Each child tag of the `rtvLayout` tag must specify the `region` attribute with a value of `north`, `south`, `east`, `west`, or `center`. This determines the location of each panel in the display.

Typically, an `rtvLayout` tag contains one of the following combinations:

- A main `rtvDisplayPanel` tag whose `region` attribute is set to `center`.

  An `rtvAccordionPanel` tag or an `rtvTreePanel` tag whose `region` attribute is set to `west` or `east`.

  Possibly other secondary `rtvDisplayPanel` tags with other `region` attribute values.

- A main `rtvTabbedDisplayPanel` tag whose `region` attribute is set to `center`.

  Possibly other secondary `rtvDisplayPanel` tags with other `region` attribute values.

An `rtvLayout` tag can contain the following attributes:

- `dividers` - Set to true if you want a divider to be drawn between child panels. The default is false.
- `title` - Specify the title of the main window.

As a child of the `rtvLayout` element, you can specify one or more `rtvDisplayPanel` elements. An `rtvDisplayPanel` element creates a panel. The display inside the panel is specified by the following `rtvDisplayPanel` attributes:

- `display` - Specify the location of this CardPanel if it is in a BorderPanel. Valid values are `west`, `east`, `center`, `north`, and `south`
- `name` - Specify the **Window Name** previously specified in the **Drill Down Properties** dialog. If you are using tabbed panels and you do not specify a name, it is constructed by using the display name and substitutions to make it easy to drill down between tabs. In this case, when you drill down from a tab by using the Current Window option and the specified display with the specified substitutions is already loaded in another tab, the Display Viewer switches to that tab.
- `region` - Specify the position of the panel as `west`, `east`, `center`, `north`, or `south`.
- `subs` - Specify initial substitutions for this panel. Substitutions are optional and must use the following syntax:

  ```
  $subname:subvalue $subname2:subvalue2
  ```

  If a substitution value contains a single quote you must escape it by using a forward slash, for example:

  ```
  $filter:Plant=/'Dallas/'
  ```

  If a substitution value contains a space it must be enclosed in single quotes. Do not escape these single quotes. Following is a correct example:

  ```
  $subname:subvalue $subname2:'sub value 2'
  ```

  A substitution string cannot contain the following characters:

  ```
  : | . tab space , ; = <  > ' " & / \ { } [ ] ( )
  ```

  Substitutions that you set in Application Options apply to all displays.

Following is an example of; an `rtvDisplayPanel` element:

```
<rtvDisplayPanel region="north" name="title_panel" display="title.rtv"
 </rtvDisplayPanel>
```

#### *Configuring panels with accordion controls*

As a child of the `rtvLayout` element, you can specify one or more `rtvAccordionPanel` elements. An `rtvAccordionPanel` element creates a panel that contains an accordion control for display navigation. The accordion control assumes there is a panel in the center region that was created with the `rtvDisplayPanel` element. The accordion control sends its navigation commands to this center panel.

The contents of a panel created with the `rtvAccordionPanel` element cannot be more than two levels deep, not including the root node. If you require deeper nesting create a panel with the `rtvTreePanel` element.

Use the following attributes to specify the location of an accordion control panel:

- `region` - Specify the position of the panel as `west`, `east`, `center`, `north`, or `south`. The default is `center`.
- `width` - Specify the width of the panel in pixels. The default is 125.

### *Configuring static tree navigation panels*

As a child of the `rtvLayout` element, you can specify one or more `rtvTreePanel` elements. An `rtvTreePanel` element creates a panel that contains a static navigation tree. The navigation tree assumes there is a panel in the center region that was created with the `rtvDisplayPanel` element. The static navigation tree sends its navigation commands to this center panel.

There are two ways to create a tree-driven, multi-panel application: the static tree navigation panel and the tree control. Use a static tree navigation panel when you know the specific sources that are to populate the tree and they remain constant for the life of the application. For example, if you know all the displays that compose your application and the static representation of a tree will be used only for navigating those displays the static tree navigation panel is suitable, as well as easier to configure.

Use the tree control when the number of tree nodes, leaves, labels, or icons changes during the lifetime of the application. Data can be provided that will change the nodes and leaves of the tree and also change the label and icon representation on the tree with dynamic data. See "Using tree controls in panel displays" on page 193.

Use the following attributes to specify the location of a static tree navigation panel:

- `region` - Specify the position of the panel as `west`, `east`, `center`, `north`, or `south`. The default is `center`.
- `width` - Specify the width of the panel in pixels. The default is 125.

### *Configuring tabbed navigation panels*

As a child of the `rtvLayout` element, you can specify one or more `rtvTabbedDisplayPanel` elements. An `rtvTabbedDisplayPanel` element creates a panel with tabs for navigation. The display inside the panel is specified by the following `rtvTabbedDisplayPanel` attributes:

- `tabs` - Specify the name of a tab definition file. This XML file should describe the tabs you want in the panel. See "Using tab definition files" on page 192.
- `display` - Specify the name of the display (`.rtv`) file to load into the panel.
- `subs` - Specify initial substitutions for this panel. Substitutions are optional and must use the following syntax:

  ```
  $subname:subvalue $subname2:subvalue2
  ```

  If a substitution value contains a single quote you must escape it by using a forward slash, for example:

  ```
  $filter:Plant=/'Dallas/'
  ```

  If a substitution value contains a space it must be enclosed in single quotes. Do not escape these single quotes. Following is a correct example:

  ```
  $subname:subvalue $subname2:'sub value 2'
  ```

  A substitution string cannot contain the following characters:

  ```
  : | . tab space , ; = <  > ' " & / \ { } [ ] ( )
  ```

Substitutions that you set in Application Options apply to all displays.

- `region` - Specify the position of the panel as `west`, `east`, `center`, `north`, or `south`.
- `placement` - Specify `top` or `bottom` to indicate where you want the tabs to appear in the panel.

Following is an example of; an `rtvDisplayPanel` element:

```
<rtvDisplayPanel region="north" name="title_panel" display="title.rtv"
 </rtvDisplayPanel>
```

#### *Using tab definition files*

When you specify an `rtvTabbedDisplayPanel` element in a panels configuration file, you must set the element's `tabs` attribute to the name of the tab definition file that defines the tabs you want in the panel.

The tab definition file must start with the following:

```
<?xml version="1.0" ?>
<navtree>
```

The tab definition file must end with the following:

```
</navtree>
```

For example:

```
<?xml version="1.0" ?>
<navtree>
 <node label="Bar Chart" display="disp1.rtv"/>
 <node label="History Graph" display="disp2.rtv" subs="$v1:xyz"/>
</navtree>
```

Inside the `navtree` element, you can define one or more `node` elements. Each `node` element adds a tab to the panel. You can specify the following attributes for each `node` element:

- `display` - Specify the name of the display (`.rtv`) file.
- `label` - Specify the label for this tab in the panel.
- `subs` - Specify substitutions to apply to this tab. Substitutions are optional and must use the following syntax:

  ```
  $subname:subvalue $subname2:subvalue2
  ```

  If a substitution value contains a single quote you must escape it by using a forward slash, for example:

  ```
  $filter:Plant=/'Dallas/'
  ```

  If a substitution value contains a space it must be enclosed in single quotes. Do not escape these single quotes. Following is a correct example:

  ```
  $subname:subvalue $subname2:'sub value 2'
  ```

  A substitution string cannot contain the following characters:

  ```
  : | . tab space , ; = <  > ' " & / \ { } [ ] ( )
  ```

#### *Examples of configuration files for multiple panels*

The following `PANELS.ini` file uses the new tags and creates a title panel at the top, an accordion panel on the left, and a main display in the center. There are draggable dividers between all panels.

```
<?xml version="1.0" ?>
<panels xmlns="www.sl.com" version="1.0">
<rtvLayout title="Accordion Example" dividers="true">
    <rtvDisplayPanel region="north" name="title_panel"
            display="title.rtv"/>
    <rtvAccordionPanel region="west" width="200" navdata="navtree.xml"/>
    <rtvDisplayPanel region="center" name="main_panel"
            display="chart_main.rtv"/>
</rtvLayout>
</panels>
```

The next `PANELS.INI` file creates a tabbed display panel at the top and a title panel at the bottom.

```
<?xml version="1.0" ?>
<panels xmlns="www.sl.com" version="1.0">
<rtvLayout title="Tab Example">
  <rtvTabbedDisplayPanel region="center" tabs="navtabs.xml"
    display="stock_chart"/>
  <rtvDisplayPanel region="south" name="title_panel" display="title.rtv"/>
</rtvLayout>
</panels>
```

## Using tree controls in panel displays

The tree control (class name: `obj_c1tree`) lets you create a rich and compact visual presentation of hierarchical data. This control is most often used in a multi-panel application for display navigation. The control tree can also be used in any application where hierarchical data is most effectively displayed using expandable/collapsible tree nodes.

There are two methods for creating a tree-driven multi-panel application:

- Static tree navigation panel — Use a static tree navigation panel when you know the specific sources that will populate the tree and they remain constant for the life of the application. For example, if you know all the displays that compose your application and the static representation of a tree will be used only for navigating those displays, the static tree navigation panel is suitable (and is easier to configure). To configure the static tree navigation panel, add the tree using the `rtvTreePanel` tag to the `PANELS.ini` file. For details about configuring the tree, see "Configuring static tree navigation panels" on page 191.
- Tree control — Use the tree control method if the number of nodes or leaves, labels or icons of the tree change during the lifetime of the application. Data can be provided that will change the nodes and leaves of the tree and also change the labels, and icon representations on the tree with dynamic data.

When using the tree control to construct an application with multiple panels one panel displays a `.rtv` file that has instanced the tree control and the other contains the displays which are drilled down to by selecting items on the tree.

The following illustrates a two-panel application in which the tree control in the left panel updates the display in the right panel:


You can optionally configure tree control icons, using images of your choice, to visually indicate the type of elements in the tree, for example, Production or Sales, whether the element is in a critical state, and to also propagate the status of priority elements to the top of the tree. See "Configuring tree control icons" on page 200.

### *Creating tree controls*

The input of tabular data determines the content of the tree control, as well as the appearance of each object in the tree. As with other controls, to configure a drill-down, set substitutions, or run a command when a user clicks a tree node, use the `actionCommand` property. As with other table-driven objects, the `drillDownColumnSubs` property can be configured to set substitutions to column values from the row in the table (attached to the `valueTable` property) that corresponds to the selected tree node.

After you attach your tabular data to the tree control `valueTable` property, specify the table format for the tree in the `valueTableFormat` property. The table format is determined by the format of the table you attach to the `valueTable` property. There are two table format options, each with their own requirements:

- Row-leaf: This format is intended for use when the `valueTable` property is attached to a table and all leaves in the tree are at the same depth. For example, where the tree control is attached to a scenario instance table. The `nodeIndexColumnNames` property specifies the columns from the scenario instance table that will appear in the hierarchy in the tree control.
- Row-node: If the row-leaf format is not suitable for your data, use the row-node format. Your data table must also contain a row for each node in the tree, including the top-level node (rather than just the leaf nodes, as with the row-leaf format),

as well as a column for the node and a column for the parent node. The row-node format allows each leaf of the tree to have a different depth.

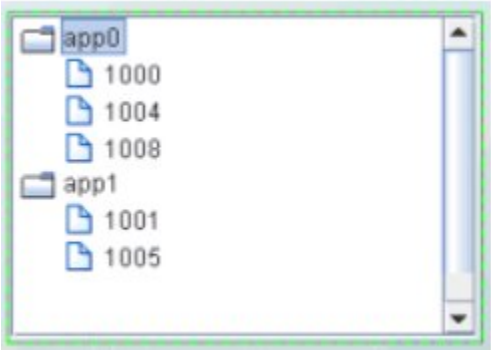
The default table format is row-leaf. The following are examples of the row-leaf and row-node table formats, which both produce the tree in the image that follows. Here is a row-leaf table:

| App Name | PID |
|---|---|
| App0 | 1000 |
| App0 | 1004 |
| App0 | 1008 |
| App1 | 1001 |
| App1 | 1005 |

Here is a row-node table:

| Node | Parent |
|---|---|
| app0 | |
| 1000 | app0 |
| 1004 | app0 |
| 1008 | app0 |
| app1 | |
| 1001 | app1 |
| 1005 | app1 |

Here is the tree control that both these tables produce:




After you configure the tree table format, you can optionally configure the tree control icons. See "Configuring tree control icons" on page 200.

#### Creating row-leaf format control trees

In the row-leaf table format, there is one row in the table for each leaf node in the tree. A leaf node is added to the tree for each row in the table attached to the `valueTable` property. The path to a leaf node (that is, the ancestor nodes of the leaf) is determined by the values in each of the table columns specified by the `nodeIndexColumnNames` property. When the `valueTable` property is attached to the scenario instance table, the tree's `nodeIndexColumnNames` property is typically set to the same columns that are specified in the **Display variables** field of the **Attach to Apama** dialog.

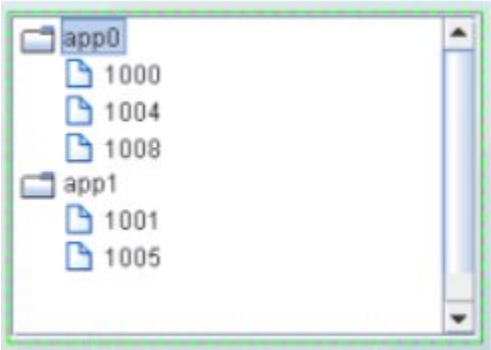
To illustrate how to create a tree using the row-leaf format, consider a table that has two columns, `App Name` and `PID`, and the following rows:

| App Name | PID |
|---|---|
| app0 | 1000 |
| app0 | 1004 |
| app0 | 1008 |
| app1 | 1001 |
| app1 | 1005 |

Set tree control properties as follows:

- Attach the tree control object's `valueTable` property to Apama as you would attach any table object. In the **Attach to Apama** dialog, in the **DisplayVariables** field, select the variables **App Name** and **PID**.
- Set the `valueTableFormat` property to the `Row-Leaf` format.
- Set the `nodeIndexColumnNames` property to `App Name;PID`.

The following image illustrates the structure of the tree. There are two nodes labeled `app0` and `app1`. Node `app0` has three child nodes labeled `1000`, `1004`, `1008`. Node `app1` has two child nodes, labeled `1001` and `1005`.

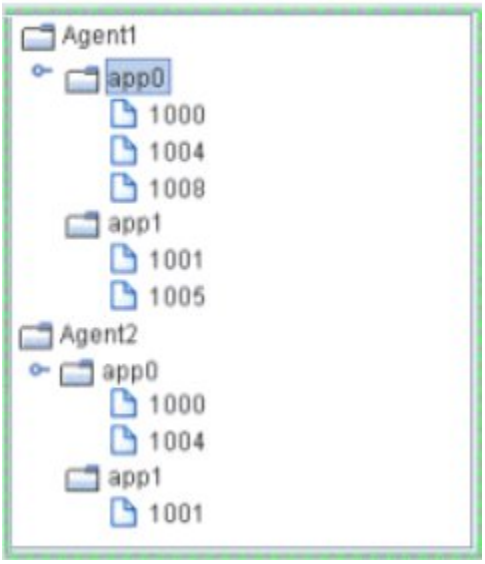



Suppose we add another column, `AgentName`, by selecting that variable from the **Display Variables** field of the **Attach to Apama** dialog. The table has the following rows:

| AgentName | App Name | PID |
|---|---|---|
| Agent1 | App0 | 1000 |
| Agent1 | App0 | 1004 |
| Agent1 | App0 | 1008 |
| Agent1 | App1 | 1001 |
| Agent1 | App1 | 1005 |
| Agent2 | App0 | 1000 |
| Agent2 | App0 | 1004 |
| Agent2 | App1 | 1001 |

We also update the tree control `nodeIndexColumnNames` property to **AgentName;App Name;PID**.

The following figure illustrates the new structure of the tree. The tree now has two top-level nodes labeled **Agent1** and **Agent2**, each with two child nodes, **app0** and **app1**.




As illustrated above, the label string for a node at depth `N` is taken from the `N`th column in the `nodeIndexColumnNames` property. Therefore, the labels for the top-level nodes come from the first column in the `nodeIndexColumnNames` property (`AgentName`), the labels for the second-level nodes come from the second column in `nodeIndexColumnNames` property (`App Name`), and so forth.

To specify node labels from a different set of `valueTable` columns, use the `nodeLabelColumnNames` property. Enter a semicolon-separated list of column names in the `nodeLabelColumnNames` property, one for each level in the tree, where the `N`th column name in the list contains the labels for tree nodes at depth `N`.

To modify tree control icons or configure tree control icon behavior, see "Configuring tree control icons" on page 200.

#### Creating row-node format tree controls

In the row-node format tree control, there is one row in the table for each node in the tree, including the top-level node rather than just one row for each of the leaf nodes as with the row-leaf format.

There are two columns in the table that help define the tree structure:

- One of the table columns contains a node ID string and is identified by the `nodeIdColumnName` property. By default, the node ID string is used as the node label in the tree. The node ID string must be unique among all nodes with the same parent. Or, if the `uniqueNodeIdFlag` property is checked, each node ID string must be unique in the entire tree.
- Another table column contains the ID of the parent node, which is identified by the `parentIdColumnName` property.

To create the same tree as the row-leaf format example in the previous topic, a table representation of the tree control using the row-node format would be as follows:

| Node | Parent |
|---|---|
| app0 | <blank> |
| 1000 | app0 |
| 1004 | app0 |
| 1008 | app0 |
| app1 | <blank> |
| 1001 | app1 |
| 1005 | app1 |

The <blank> entries represent an empty string, which indicates that nodes app0 and app1 have no parent, making them top-level nodes in the tree.

Set the tree control properties as follows:

- valueTable property to attach to the table that contains the data to be displayed
- valueTableFormat property to the Row-Node format
- nodeIdColumnName property to Node
- parentIdColumnName property to Parent

The result is a tree structure with two top-level nodes labeled app0 and app1. Node app0 has three child nodes labeled 1000, 1004, 1008. Node app1 has two child nodes, labeled 1001 and 1005.

To add another tree level for the AgentName, as in the Row-Leaf format example in the previous topic, modify the table as follows:

| Node | Parent |
|---|---|
| Agent1 | <blank> |
| app0 | Agent1 |
| 1000 | app0 |
| 1004 | app0 |
| 1008 | app0 |

| Node | Parent |
|---|---|
| Agent2 | <blank> |
| app0 | Agent2 |
| 1000 | app0 |
| 1004 | app0 |
| app1 | Agent2 |
| 1001 | app1 |
| 1005 | app1 |

Also, uncheck the `uniqueNodeIdFlag` property to allow for node IDs that are not unique, such as the `app0` and `1000` nodes in the example, which are used in multiple tree levels. Because some of the rows are also identical, the order of the table rows is important. For example, this row appears twice in the table: `1000app0`. In each case the `1000app0` row comes after a unique parent row: first under `app0Agent1` and then under `app0Agent2`. The tree uses this information to determine where to place the node for `1000` in each case.

The tree now has two top-level nodes labeled `Agent1` and `Agent2`, each with two child nodes, `app0` and `app1`.

By default, the node ID string is used as the node label in the tree. If a different column in the table must provide the label, specify the name of that column in the `nodeLabelColumnName` property.

In the row-node format, each branch of the tree can have a different depth, while in the row-leaf format all branches typically have the same depth, which is the number of columns specified in the `nodeIndexColumnNames` property.

To modify tree control icons or configure tree control icon behavior, see "Configuring tree control icons" on page 200.

#### *Configuring tree control icons*

You can optionally configure tree control icons, using images of your choice, to visually indicate the type of elements in the tree, for example, Production or Sales, whether the element is in a critical state, and to also propagate the status of priority elements to the top of the tree.

The use of one or both of the following icons is optional. Each node in the tree control can display these two configurable icons:

- Type icon — Use the type icon to assign static images to nodes that indicate either the type of element it contains, or its level in the tree. By default, a folder image is used for all non-leaf nodes, and a document image is used for all leaf nodes.

  For example, if you have groups of elements based on geographic location, you could assign an icon for the country, state and city (for example, US, California, San Francisco). Or, if you have groups of elements based on their function, you could assign an icon for each function (Purchases, Operations, Sales, and so forth). You can also assign images for each depth in the tree for a visual indication of where you are while navigating within the tree.

- Status icon — Use the status icon to assign images that are used when an element in the tree has a specified value. You can also configure the status in order of priority so that the most critical status is propagated up the tree first. By default, there is no status icon.

If a node has a type icon and a status icon, the type icon always appears to the left of the status icon.

#### Attaching a tree control icon to data

For convenience, both the type icon and the status icon can be attached to data. The type icon and status icon have different data table requirements. Typically, an attachment to a static XML file containing the appropriate tables is used. The following describes the data table format requirements:

- Type icon — To attach the type icon to data, use the `nodeTypeProperties` property. The data attachment must be a two-column table. Typically, a static XML file containing the table is used. The first column must contain string values of `_node` (for non-leaf nodes), `_leaf` (for leaf nodes), numeric values for depth, or string values that match the node labels, or the values from the column in `valueTable` specified by the `nodeTypeColumnName` property. The second column must be the path to the `.png`, `.gif`, or `.jpg` image. The default assignments are `_node`, `rtvTreeNode16.png` and `_leaf`, `rtvTreeLeaf16.png`. The column names are not important.

- Status icon — To attach the status icon to data, use the `nodeStatusProperties` property. The data attachment must be a three-column table. Typically, a static XML file containing the table is used. The first column must contain string values that match values from the column in `valueTable` specified by the `nodeStatusColumnName` property. The second column must be the path to the `.png`, `.gif`, or `.jpg` image. The third column must contain the non-negative integer priority value.

A static XML file is read once each time you run Dashboard Viewer. If you specify (or modify) an XML source using the **Application Options** dialog, you may specify whether that XML source is static. For details, see Creating XML Sources.

### Configuring tree control type icons

Type icons indicate the type of node in the tree. The type icon for each node is determined either by the value of a column in the `valueTable` property, or by the node position in the tree. By default, the type icon appears to the left of the node label.

This section describes how to configure type icons using the **Node Properties** dialog. You can also configure type icons by attaching the `nodeTypeProperties` property to data.

To configure the type icon, use the `nodeTypeProperties` property to define a two-column table of data. Select the `nodeTypeProperties` property in the property sheet, then click the  button to open the **Node Properties** dialog.

In the **Node Properties** dialog, the **Node Depth or Type** column lists the available nodes. The first two rows, **non-leaf node** and **leaf node**, indicate the default settings: non-leaf nodes in the tree use a folder image and leaf nodes use a document image. To change the default setting, click the  button in the **Image** column for the node and choose an icon from the **Select Image** dialog.

The next five rows, numbered **0 - 4**, represent the node depth or level, zero (**0**) being the root node. The **Image** column lists the icons being used for each node. By default, the **Image** column for all of those rows is `<blank>`, indicating that the **non-leaf node** and **leaf node** icon assignments are used. Icon assignments override **non-leaf node** and **leaf node** assignments.

You can also assign an image icon based on node level. Such an icon provides a visual indication of where you are while navigating in the tree. To assign an image to a specific node level in the tree, click the  button for one of the rows numbered **0 - 4** in the **Image** column and choose an icon from the **Select Image** dialog. Repeat for each node level.

You can assign an image icon based on node labels you create that describe the nodes as a group. For example, suppose the **Node Depth or Type** column contains the string **Davies** with the  image selected.

This means that all nodes whose label matches the **Davies** string display the  image in the tree.

To assign an image to a specific node type in the tree, assign a column name in the **nodeTypeColumnName** property. Select the **nodeTypeProperties** property in the property sheet, then click on the button to open the **Node Properties** dialog. Click **New** to add a custom row to the table. A drop-down list of values for the column assigned to the **nodeTypeColumnName** property appears in the **Node Depth or Type** column. Select a column name from the drop-down list. Click the button in the **Image** column and choose an icon (to use for all nodes that have that column name in the **valueTable** row that corresponds to the node) from the **Select Image** dialog.

You can also type a string in the **Node Depth or Type** column and the **Image** column.

To not use an icon, in the **Node Properties** dialog, select the icon in the **Image** column, then click **Clear**.

Note that the root node is invisible if the **rootNodeLabel** property is blank.

#### Configuring tree control status icons

Status icons indicate the current state of a node. You can configure the status icon to propagate the status of a child node up the tree to its ancestors. The status icon shown for an ancestor node corresponds to the current highest status priority among all of its descendants.

The status icon for a node is determined by the discrete value of a specified column in the `valueTable` property. The values can be strings or numbers. The comparison is done for an exact match. If the current status value for a tree node does not match any of the values you specify in the `nodeStatusProperties` property, no status icon is displayed for that node.

By default, the status icon appears on the left of the node label. The value, `Left` or `Right`, is specified in the `nodeStatusIconPos` property. If a node has both a type icon and a status icon, the type icon always appears to the left of the status icon. By default, no status icons appear in the tree.

This section describes how to configure status icons using the **Node Properties** dialog. You can also configure status icons by attaching the `nodeStatusProperties` property to data. For details about that, see "Attaching a tree control icon to data" on page 201.

To configure status icons, specify the status table column name in the `nodeStatusColumnName` property, then use the `nodeStatusProperties` property to define a three-column table of data and configure icon behavior. The `nodeStatusProperties` property is visible only if the `nodeStatusColumnName` property is non-blank.

Select the `nodeStatusProperties` property in the property sheet, then click the  button to open the **Node Properties** dialog.

In the **Node Properties** dialog, to map an image to a node status, click **New**, then click in the **Status Value** column. A drop-down list appears containing all values in the node status column of the `valueTable` property (which you previously specified in the `nodeStatusColumnName` property). Select a value from the drop-down list.

Click the  button in the **Image** column for the node and choose an icon from the **Select Image** dialog. This icon is used as the status icon for all nodes that match the value selected in the **Status** column.

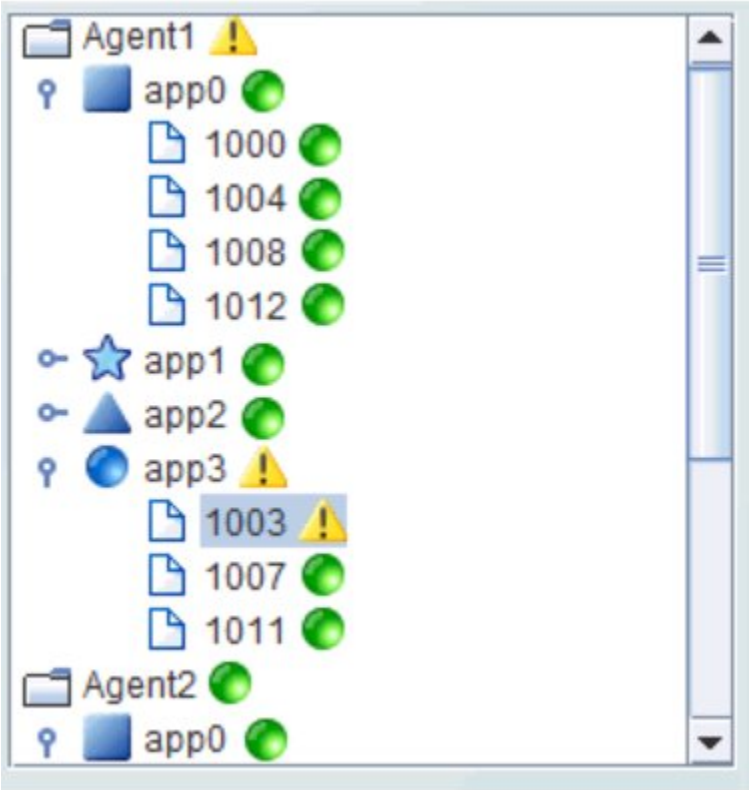
Click the  button in the **Priority** column for the node and assign an integer value: **0**, **1**, **2**, **3**, **4**, **5**, or a higher number. There is no upper limit on the number. The largest number is the highest priority and is propagated up the tree first. A value of zero (0) is not propagated. You might want to assign a value of zero to multiple nodes so that they do not propagate up the tree. A non-zero value can be assigned only once.

For example, suppose the `nodeStatusColumnName` property is set to the `Application Status` table column of the `valueTable` property. You could define the mapping for the `nodeStatusProperties` property as follows:

| Status Value | Image | Priority |
|---|---|---|
| **Blocked** |  | **2** |
| **Running** |  | **1** |

The values in the **Application Status** column of each row in the `valueTable` property is compared to the two values listed in the **Status Value** column (**Blocked** and **Running**). If the **Application Status** value in one of the rows is **Blocked**, the  status icon is displayed as the status icon for that row. If there is no match, for example, the **Application Status** value in one of the rows is unknown, no status icon is displayed in the tree node for that row. Because the  status icon is assigned the highest priority, the  status icon is always propagated up the tree first. If none of the rows has a **Blocked** status, the  status icon is propagated up the tree.

For example, in the following image, the priority status of a single node, **app3/1003**, is propagated up to its parent, **app3**, and also to the top-level ancestor, **Agent1**.



### *Specifying tree control properties*

There are a number of properties that you can specify for a tree control.

#### Specifying tree control background properties

The `bgColor` property determines the color of the tree control background. Select the **bgColor** property and click  . Choose a color from the palette to set the background color of the tree control.

#### Specifying tree control data display properties

The following properties specify how data is displayed in the tree control.

- **nodeIdColumnName**

  This property is available when the **valueTableFormat** is **Row-Node**. With the **Row-Node** format there are two table columns that define the tree structure: the **nodeIdColumnName** property and the **parentIdColumnName** property.

  The **nodeIdColumnName** property specifies the table column containing the node ID string. The node ID string must be unique among all nodes with the same parent. Or, if the **uniqueNodeIdFlag** property is checked, each node ID string must be unique in the entire tree. By default, the node ID string is used as the node label in the tree.

- **nodeIndexColumnNames**

  This property is available when the **valueTableFormat** is **Row-Leaf**. It specifies the path to a leaf node, that is, the ancestor nodes of the leaf.

  When the **valueTable** property is attached to the current table of a scenario instance the `nodeIndexColumnNames` property is typically set to the same columns that are specified in the **Display variables** field of the **Attach to Apama** dialog used to set the `valueTable` property.

  Enter a semicolon-separated list of column names, where the Nth column name in the list contains the labels for tree nodes at depth N. The labels for top-level nodes are defined by the first column in the **nodeIndexColumnNames** property, the labels for the second-level nodes are defined by the second column, and so forth. For example:

  **AgentName;App Name;PID**

  The labels for the top-level nodes are defined by the **AgentName** column, the labels for the second-level nodes are defined by the **App Name** column, and labels for the third-level nodes are defined by the **PID** column.

  To specify node labels from a different set of **valueTable** columns, use the **nodeLabelColumnNames** property.

- **nodeLabelColumnName**

  This property is available when the **valueTableFormat** is **Row-Node**. By default, the node ID string is used as the node label in the tree. Use the **nodeLabelColumnName** property to specify a different **valueTable** column to provide the label.

- **nodeLabelColumnNames**

  This property is available when the **valueTableFormat** is **Row-Leaf**. Use the **nodeLabelColumnNames** property to specify a different set of **valueTable** columns to provide node labels. Enter a semicolon-separated list of column names, one for each level in the tree, where the Nth column name in the list contains the labels for tree nodes at depth N.

- **nodeStatusColumnName**

  This property applies to the status icon. It specifies the name of the **valueTable** column containing node status values. The column specified populates the **Node Properties** dialog **Status Value** column, in which you map node status values to image icons. The icons are displayed for any node whose value matches the value selected.

- **nodeTypeColumnName**

  This property applies to the type icon. It specifies the name of the **valueTable** column containing values to use for mapping icon images to node types in the tree. The column specified populates the list of available values in the **Node Properties** dialog **Node Depth or Type** column, in which you map node types to image icons. The icons are displayed for any node whose value matches the value selected.

- **parentIdColumnName**

  This property is available when the **valueTableFormat** is **Row-Node**. With the **Row-Node** format there are two table columns that define the tree structure: the **parentIdColumnName** property and the **nodeIdColumnName** property.

  The **parentIdColumnName** property specifies the table column containing the parent node ID.

- **uniqueNodeIdFlag**

  This property is available when the **valueTableFormat** is **Row-Node**.

  When enabled, this property specifies that each node ID string must be unique in the entire tree. When disabled, it specifies that each node ID string must be unique among all nodes with the same parent.

- **valueColumnName**

  Specifies the name of the column whose value is assigned to the **$value** variable when a node in the tree is selected. If not specified, the label string of the selected node is assigned to the **$value** variable. The **$value** variable is the only substitution that can be used in the **Display Name** field of a drill-down command.

- **valueTable**

  Attach your tabular input data to this property. There are two **valueTable** format options, each with their own requirements: **Row-Leaf** and **Row-Node**.

  As with other table-driven objects, the **drillDownColumnSubs** property can be configured to set substitutions to column values from the row in the **valueTable** that corresponds to the selected tree node.

- **valueTableFormat**

  Specifies the format of the **valueTable**: **Row-Leaf** or **Row-Node**.

- **varToSet**

  Allows you to update the attached variable with the value from the control.

#### Specifying tree control interaction properties

The following properties specify interactions in the tree control.

- **actionCommand**

  Use the **actionCommand** property to assign a command to the tree. You can configure the tree to open a drill-down display, set substitutions, or execute a command in response to a user click on a tree node.

  The **actionCommand** property can reference the value from the tree by using the keyword **$value**. When the command is executed, the variable attached to the **varToSet** property is updated with the selected node data.

  The **drillDownColumnSubs** property can be configured to set substitutions to column values from the row in the **valueTable** that corresponds to the selected tree node.

  If the **execOnLeafOnlyFlag** property is checked, the tree **actionCommand** property executes only when a leaf node is clicked (a click on a non-leaf node expands only the node). If unchecked, the tree **actionCommand** property executes on all nodes, not just the leaf.

- **commandCloseWindowOnSuccess**

  If selected, the window that initiates a system command will automatically close when the system command is executed successfully. This property only applies to system commands.

  With data source commands, the window is closed whether or not the command is executed successfully.

  For multiple commands, this property is applied to each command individually. Therefore, if the first command in the multiple command sequence succeeds, the window will close before the rest of the commands are executed.

  The **commandCloseWindowOnSuccess** property is not supported in the Display Server.

- **commandConfirm**

  If selected, the command confirmation dialog is enabled. Use the **commandConfirmText** property to write your own text for the confirmation dialog, otherwise text from the command property will be used.

  For multiple commands, if you confirm the execution then all individual commands will be executed in sequence with no further confirmation. If the you cancel the execution, none of the commands in the sequence will be executed.

- **commandConfirmText**

Enter command confirmation text directly in the **Property Value** field or select the  button to open the **Edit commandConfirmText** dialog. If **commandConfirmText** is not specified, then text from the command property will be used.

- **drillDownColumnSubs**

  Use the **drillDownColumnSubs** property to set substitutions to column values from the row in the **valueTable** that corresponds to the selected tree node.

  Select the  button to open the **Drill Down Column Substitutions** dialog to customize which substitutions are passed into drill-down displays.

- **enabledFlag**

  If unchecked, the tree nodes are the color gray and do not respond to user input.

- **execOnLeafOnlyFlag**

  If checked, the tree **actionCommand** is executed only for leaf nodes, and a click on a non-leaf node only expands the node. Also, the mouseover tooltip only appears for leaf nodes.

  If unchecked, the tree **actionCommand** property executes on all nodes, and the mouseover tooltip appears for all nodes.

- **mouseOverFlag**

  Specifies whether a tooltip appears when the cursor is positioned over a node. The tooltip shows the node path (the node label preceded by the labels of all of its ancestors), the node status (if the **nodeStatusColumnName** property is specified), and its value (if the **valueColumnName** property is specified).

- **tabIndex**

  Use the **tabIndex** property to define the order in which the tree receives focus when navigated from your keyboard. Initial focus is given to the object with the smallest **tabIndex** value, from there the tabbing order proceeds in ascending order. If multiple objects share the same **tabIndex** value, initial focus and tabbing order are determined by the alpha-numeric order of the table names. Tables with a **tabIndex** value of **0** are last in the tabbing order.

  The **tabIndex** property does not apply to tables in the Display Server, nor to objects that are disabled, invisible, or have a value of less than 0.

#### Specifying tree control label properties

The following properties specify the appearance of tree control labels.

- **labelTextColor**

  This property sets the color of label text. Click the  button and choose a color from the palette.

- **labelTextFont**

This property sets the font of label text. Select the font from the drop-down menu.

- **labelTextSize**

  This property sets the height of the label in pixels.

#### Specifying tree control node structure properties

The following properties specify the node structure in the tree control.

- **nodeStatusIconPos**

  Specify the status icon position in the tree: **Left** or **Right**. By default, the status icon appears on the left of the node label. If a node has both a type icon and a status icon, the type icon always appears to the left of the status icon. By default, no status icons appear in the tree.

- **nodeStatusProperties**

  This property specifies the status icon for a node. By default, no status icon is displayed.

  Click the  button to open the **Node Properties** dialog and map images to values, and set the status priority order for propagation up the tree.

  The **nodeStatusProperties** property is visible only if the **nodeStatusColumnName** property is non-blank.

  You can also use the **nodeStatusProperties** property to attach a status icon to data. The data attachment must be a three-column table. Typically, a static XML file containing the table is used. The first column must contain string values that match values from the column in the **valueTable** specified by the **nodeStatusColumnName** property. The second column must be the path to the `.png`, `.gif`, or `.jpg` image. The third column must contain the non-negative integer priority value.

  A static XML file is read once each time you run Dashboard Viewer. If you specify (or modify) an XML source using the **Application Options** dialog, you may specify whether that XML source is static.

- **nodeTypeProperties**

  This property specifies the type icon for a node. By default, non-leaf nodes in the tree use a folder image and leaf nodes use a document image.

  Click the  button to open the **Node Properties** dialog to map images to nodes. Mapping can be based on the node depth in the tree or the type of node.

  You can also use the **nodeTypeProperties** property to attach a type icon to data. The data attachment must be a two-column table. Typically, a static XML file containing the table is used. The first column must contain string values of **_node** (for non-leaf nodes), **_leaf** (for leaf nodes), numeric values for depth, or string values that match the node labels, or the values from the column in the **valueTable** specified by the **nodeTypeColumnName** property. The second column must be the path to the `.png`,

`.gif`, or `.jpg` image. The default assignments are **_node**, **rtvTreeNode16.png** and **_leaf**, **rtvTreeLeaf16.png**. The column names are not important.

The logic for determining which type icon is used is as follows.

If the **nodeTypeColumnName** property specifies column **C**, and the value of **C** in the **valueTable** row that corresponds to **N** is **V**, and there is a row in **nodeTypeProperties** that assigns value **V** to image **I1**, then **I1** is used as the type icon for **N**. Otherwise:

- If the label of node **N** is **XYZ**, and there is a row in the **nodeTypeProperties** property that assigns value **XYZ** to image **I2**, then **I2** is used. Otherwise,
- If the depth of node **N** is **D**, and there is a row in the **nodeTypeProperties** property that assigns depth **D** to image **I3**, **I3** is used. Otherwise,
- If **N** is a leaf, and the leaf node image is **I4**, **I4** is used. If **I4** is blank no type icon appears. Otherwise,
- If the non-leaf node image is **I5**, **I5** is used. If **I5** is blank no type icon appears.

A static XML file is read once each time you run Dashboard Viewer. If you specify (or modify) an XML source using the **Application Options** dialog, you may specify whether that XML source is static.

- **rootNodeLabel**

  Specify whether the tree root node is visible. By default, this property is blank and the root node is not visible.

#### Specifying tree control object layout properties

The following properties specify the layout in the tree control.

- **anchor**

  Specifies where to anchor an object in the display. If an object has the **dock** property set, the **anchor** property will be ignored.

  The **anchor** property is only applied when the dimensions of the display are modified, either by editing **Background Properties** or resizing the window in **Layout** mode.

  Select **None**, or one or more the following options:

  - **None** - Object not anchored. This is the default.
  - **Top** - Anchor top of object at top of display.
  - **Left** - Anchor left side of object at left of display.
  - **Bottom** - Anchor bottom of object at bottom of display.
  - **Right** - Anchor right side of object at right of display.

  When a display is resized the number of pixels between an anchored object and the specified location remain constant. If an object is anchored on opposite sides (that is, top and bottom or left and right), the object will be stretched to fill the available

space. If the **Resize Mode** is set to **Scale** and an object is anchored on opposite sides, then the object will be moved rather than stretched to fill the available space.

- **dock**

  Specifies the docking location of an object in the display. An object should not be docked if the **Resize Mode** is set to **Scale**.

  Select from the following options:

  - **None** - Object is not docked. This is the default.
  - **Top**- Dock object at top of display.
  - **Left** - Dock object at left of display.
  - **Bottom** - Dock object at bottom of display.
  - **Right** - Dock object at right of display.
  - **Fill** - Dock object in available space remaining in the display after all docked objects are positioned.

  If the dimensions of the display are modified, either by editing **Background Properties** or resizing the window in **Layout** mode, the properties (**objX**, **objY**, **objWidth** and **objHeight**) of docked objects will automatically adapt to match the new size of the display.

  When multiple objects are docked to the same side of the display, the first object is docked against the side of the display, the next object is docked against the edge of the first object, and so on.

  When objects are docked to multiple sides of the display, the order in which objects were added to the display controls docking position. For example, suppose the first object added to the display is docked at the top and the second object is docked at the left. Consequently, the first object will fill the entire width of the display and the second object will fill the left side of the display from the bottom of the first object to the bottom of the display.

  Objects in a display that have the **dock** property set to **Fill**, are laid out across a grid in the available space remaining after all docked objects are positioned. By default, the grid has one row and as many columns as there are objects in the display. You can modify the grid in the **Background Properties** dialog.

  Once an object is docked, there are some limitations on how that object can be modified.

  - Docked objects cannot be dragged or repositioned using **objX** and **objY** properties.
  - Docked objects cannot be resized using the **objWidth** or **objHeight** properties. To resize you must drag on the resize handle.
  - Docked objects can only be resized toward the center of the display. For example, if an object is docked at the top, only its height can be increased by dragging down toward the center of the display.

- Docked objects set to **Fill** cannot be resized.
- Docked objects cannot be moved using **Align**. Non-docked objects can be aligned against a docked object, but a docked object will not move to align against another object.
- Docked objects are ignored by **Distribute**.

- **objHeight**

  This property is read-only. It shows the height in pixels of the object, which is set by the height of the tree display.

- **objName**

  Name given to facilitate object management by means of the **Object List** dialog. Select **Tools** > **Object List**.

- **objWidth**

  This property is read-only. It shows the width in pixels of the object, which is set by the width of the tree display.

- **objX**

  Sets the x position of the object.

- **objY**

  Sets the y position of the object.

- **visFlag**

  Sets the visibility of the object.

#### Descriptions of unique tree control property behavior

The following describes properties that behave uniquely with the tree control.

- **valueColumnName** - This property specifies the name of the column whose value should be assigned to the **$value** variable when a node in the tree is clicked. If not specified, the label string of the selected node is assigned to **$value**. Note that **$value** is the only substitution that can be used in the **Display Name** field of a drill-down command.
- **mouseOverFlag** - If this property is checked, a tooltip appears when the cursor is positioned over a leaf node. The tooltip shows the node path (the node label preceded by the labels of all of its ancestors), the node status (if the **nodeStatusColumnName** property is specified), and its value (if the **valueColumnName** property is specified).
- **execOnLeafOnlyFlag** - If this property is checked, the tree **actionCommand** property executes only when a leaf node is clicked (a click on a non-leaf node expands only the node). If unchecked, the tree **actionCommand** property executes on all nodes, not just the leaf.

- **rootNodeLabel** - This property specifies the tree root node (of which there is only one). By default, this property is blank and the root node is not visible.

***Tree control limitations***

In the **Display Viewer**, mouseover text is displayed only if the tree has focus.

In the Thin Client:

- The tree node appearance, such as spacing and fonts, might vary slightly as compared to the **Display Viewer**, and also may vary slightly between different browsers.
- A tree node cannot expand/collapse by double-clicking it. The +/- icon must be clicked.
- In Internet Explorer, nodes expand/collapse even if the tree **enabledFlag** property is unchecked. (However, the tree **actionCommand** cannot be invoked).
- In Mozilla Firefox, the horizontal scrollbar might appear and disappear after each mouse click in the tree.
- In iOS Safari (iPad), if the tree **mouseOverFlag** property is checked, a user must click a tree node twice to invoke the tree command. The first click only displays the node mouseover text.
- In iOS Safari (iPad), scrollbars will not appear in a tree control. If the tree contains more nodes than are visible, use a two-finger drag gesture inside the tree area to scroll.
- In iOS Safari, a click on the +/- icon expands/collapses the node as expected. However, if the **execOnLeafOnlyFlag** property is unchecked, the tree command is also executed.

## Using old tags to configure the panels in a window

The tags described in this topic are deprecated. They will be removed in a future release. You should change to the new tags. See "Using new tags to configure the panels in a window" on page 189.

When using the old tags in the panels-configuration file the following tags are supported. Remember that you cannot mix old tags and new tags in the same panels-configuration file.

| Tag | Description |
|---|---|
| `BorderPanel` | A border panel allows you to specify a central display and place up to four other displays to the north, south, east or west. Border panels are implemented as `javax.swing.JPanels` with a `BorderLayout` Add a `JPanel` with a border layout to the main window. See "Using border panels" on page 214. |

| Tag | Description |
|---|---|
| `CardPanel` | A card panel allows you to stack displays so that they are all active, but only one is showing. This is useful when you have a trend graph that needs to maintain data when it is not being displayed. Card panels are implemented as `javax.swing.JPanel` with a `CardLayout`. Display Server deployments do not support card panels. Add a `JPanel` with a card layout to the main window. See "Using card panels" on page 216. |
| `GridPanel` | A grid panel allows you to arrange your panels in tabs. Add a `JPanel` with a grid layout to the main window. See "Using grid panels" on page 216. |
| `TabbedPanel` | A tabbed panel allows you to arrange your panels in tabs. Add a `JTabbedPane` to the main window. See "Using tabs panels" on page 217. |
| `RTViewNavTreePanel` | A tree panel can be used inside a border panel to display a tree that is used to navigate displays in one of the border panel regions. Add a `JPanel` containing a `JTree` into a `BorderPanel`. This requires use of the `CardPanel`. See "Using the RTViewNavTreePanel tag" on page 221. |
| `RTViewPanel` | Add a panel containing the specified display into a `BorderPanel`, `CardPanel`, `TabbedPanel`, or `GridPanel`. See "Using the RTViewPanel tag" on page 222. |

### *Using border panels*

The tags described in this topic are deprecated. They will be removed in a future release. You should change to the new tags. See "Using new tags to configure the panels in a window" on page 189.

Use the `BorderPanel` tag to add a border panel to the main window. This tag supports the following attributes:

| Attribute | Description |
|---|---|
| `minWidth` | Set the minimum width for a `BorderPanel`, in pixels. The default value is `300`. The minimum height is determined by the `minWidth` and the overall aspect ratio of the panels contained in the `BorderPanel`. The `minWidth` attribute can |

| Attribute | Description |
|---|---|
| | be used to prevent the Dashboard Viewer from being resized so small that the displays in the `BorderPanel` are unreadable. |
| `title` | Set the title of the main window. |

Use `RTViewPanel` or `RTViewNavTreePanel` subelement to specify `.rtv` files for the center, north, south, east, and west panels.

Here is an example:

```
<?xml version="1.0" ?>
<panels xmlns="www.sl.com" version="1.0">
<BorderPanel title="Test of Border Panels">
  <RTViewPanel region="north" name="north_panel" display="long_panel"
    subs="$title:'North Panel'"/>
  <RTViewPanel region="center" name="center_panel" display="small_panel"
    subs="$title:'Center Panel'"/>
  <RTViewPanel region="west" name="west_panel" display="small_panel"
    subs="$title:'West Panel'"/>
  <RTViewPanel region="east" name="east_panel" display="small_panel"
    subs="$title:'East Panel'"/>
  <RTViewPanel region="south" name="south_panel" display="long_panel"
    subs="$title:'South Panel'"/>
</BorderPanel>
</panels>
```

When you create displays for use in border panels, the height and width of each display must be set in relation to the other displays. Displays in the west, east and center must all be equal in height. The width of the display in the north and south, must equal the combined width of the displays in the west, east and center. You will need to increase the width of the display in the north and south by the border width for each border that divides the west, center and east panels. To set the height and width of a display in the Dashboard Builder, select **File** | **Background Properties** and set the **Model Width** and **Model Height**. If you are using a background image for your display, create the image so that the height and width of the image are one pixel larger than the size you want the display to be.

The following shows dimensions of display (`.rtv`) files set to fit accurately in multiple display panels:

| Display Name | Display Location | Model Width | Model Height |
|---|---|---|---|
| `small_panel.rtv` | center | 320 | 240 |
| `small_panel.rtv` | east | 320 | 240 |
| `small_panel.rtv` | west | 320 | 240 |
| `long_panel.rtv` | north | 962 | 120 |

| Display Name | Display Location | Model Width | Model Height |
|---|---|---|---|
| `long_panel.rtv` | south | 962 | 120 |

### *Using card panels*

The tags described in this topic are deprecated. They will be removed in a future release. You should change to the new tags. See "Using new tags to configure the panels in a window" on page 189.

With card layout, you use the `CardPanel` element to specify a main panel and subordinate panels. Display Server deployments do not support card layout.

The `CardPanel` tag supports the following attributes:

| Attribute | Description |
|---|---|
| `region` | Set the location of this `CardPanel` if it is in a `BorderPanel`. Valid values are `west`, `east`, `center`, `north`, and `south`. |
| `title` | Set the title of the main window. |

Here is an example:

```
<?xml version="1.0" ?>
<panels xmlns="www.sl.com" version="1.0">
<CardPanel>
  <RtViewPanel title=" Main Panels " name="main" display="main_panel"/>
      <!-- The following three panels will always be kept in memory -->
  <RtViewPanel title="Panel 101" display="med_panel" subs="$title:101">
  <RtViewPanel title="Panel 102" display="med_panel" subs="$title:102">
  <RtViewPanel title="Panel 103" display="med_panel" subs="$title:103">
      <!-- All other displays will be loaded on demand -->
</CardPanel>
</panels>
```

When you create displays for use in card panels, the height and width of each display must be the same. To set the height and width of a display in the Dashboard Builder, select **File | Background Properties** and set the **Model Width** and **Model Height**. If you are using a background image for your display, create the image so that the height and width of the image are one pixel larger than the size you want the display to be.

### *Using grid panels*

The tags described in this topic are deprecated. They will be removed in a future release. You should change to the new tags. See "Using new tags to configure the panels in a window" on page 189.

Use the `GridPanel` tag to arrange panels into a specified number of rows and columns. Display Server deployments do not support grid layout.

This tag supports the following attributes:

| Attribute | Description |
|---|---|
| `columns` | Sets the number of columns in the grid. If the number of columns is not specified, it will be calculated based on the number of `RTViewPanel`s and the specified number of rows. |
| `rows` | Sets the number of rows in the grid. If the number of rows is not specified, it will be calculated based on the number of `RTViewPanel`s and the specified number of columns. |
| `title` | Set the title of the main window. |

Here is an example:

```
<?xml version="1.0" ?>
<panels xmlns="www.sl.com" version="1.0">
<GridPanel title="Test of Grid Panels" rows="0" columns="3">
  <RTViewPanel name="detail1" display="small_panel" subs="$title:'101'"/>
  <RTViewPanel name="detail2" display="small_panel" subs="$title:'102'"/>
  <RTViewPanel name="detail3" display="small_panel" subs="$title:'103'"/>
  <RTViewPanel name="detail4" display="small_panel" subs="$title:'201'"/>
  <RTViewPanel name="detail5" display="small_panel" subs="$title:'202'"/>
  <RTViewPanel name="detail6" display="small_panel" subs="$title:'203'"/>
</GridPanel>
</panels>
```

When you create displays for use in grid panels, the height and width of each display must be the same. To set the height and width of a display in the Dashboard Builder, select **File** > **Background Properties** and set the **Model Width** and **Model Height**. If you are using a background image for your display, create the image so that the height and width of the image are one pixel larger than the size you want the display to be.

### *Using tabs panels*

The tags described in this topic are deprecated. They will be removed in a future release. You should change to the new tags. See "Using new tags to configure the panels in a window" on page 189.

Use the `TabbedPanel` tag to arrange `.rtv` files into a tabbed panel. This tag supports the following attributes:

| Attribute | Description |
|---|---|
| `placement` | Set the position of the tab. Valid arguments are `left`, `right`, `top`, and `bottom`. |

| Attribute | Description |
|---|---|
| | **Note:** This argument is ignored by the Data Server. Tabs are always in the top position. |
| `preload` | Set to `false` so that only one display at a time is loaded. |
| `title` | Set the title of the main window. |

Here is an example:

```
<?xml version="1.0" ?>
<panels xmlns="www.sl.com" version="1.0">
<TabbedPanel title="Test of Tabbed Panels" placement="top">
  <RTViewPanel title="Main Panel" display="main_panel"/>
  <RTViewPanel title="Panel 101" display="med_panel" subs="$title:101"/>
  <RTViewPanel title="Panel 102" display="med_panel" subs="$title:102"/>
  <RTViewPanel title="Panel 103" display="med_panel" subs="$title:103"/>
  <RTViewPanel title="Panel 201" display="med_panel" subs="$title:201"/>
  <RTViewPanel title="Panel 202" display="med_panel" subs="$title:202"/>
  <RTViewPanel title="Panel 203" display="med_panel" subs="$title:203"/>
</TabbedPanel>
</panels>
```

When you create displays for use in tabbed panels, the height and width of each display must be the same. To set the height and width of a display in the Dashboard Builder, select **File | Background Properties** and set the **Model Width** and **Model Height**. If you are using a background image for your display, create the image so that the height and width of the image are one pixel larger than the size you want the display to be.

By default, the displays for all tabs are loaded at startup and are never unloaded. You can set to `false` the preload attribute on the TabbedPanel tag in order to change this behavior so that only the display for the first tab is loaded at startup and the display for a tab is unloaded when the user selects another tab. In other words, if preload=false, only one display at a time is loaded in a tabbed panel.

Following is an example:

```
<?xml version="1.0" ?>
<panels xmlns="www.sl.com" version="1.0">
  <TabbedPanel title="Test of Tabbed Panel" placement="top" preload="false">
  <RtViewPanel title="Table Overview" display="overview"/>
  <RtViewPanel title="Production Table" display="production_table"/>
  <RtViewPanel title="System Table" display="system_table"/>  </TabbedPanel>
</panels>
```

### *Using tree panels*

The tags described in this topic are deprecated. They will be removed in a future release. You should change to the new tags. See "Using new tags to configure the panels in a window" on page 189.

With tree panels, you define the contents of a tree-structured navigation pane by specifying an xml file (`navtree.xml` in this example) as the value of the `navtreedata` attribute in an `RTViewNavTreePanel` element:

```
<?xml version="1.0" ?>
<panels xmlns="www.sl.com" version="1.0">
<BorderPanel>
  <RTViewPanel region="north" name="title_panel" display="title_panel"/>
  <CardPanel region="center">
    <RtViewPanel title=" Overview " name="main" display="main_panel"/>
  </CardPanel>
  <RtViewNavTreePanel region="west" width="192" height="480"
     lineStyle="Angled" navtreedata="navtree.xml">
  </RtViewNavTreePanel>
</BorderPanel>
</panels>
```

The file that you specify for the `navtreedata` attribute must be in XML, and must start with the following:

```
<?xml version="1.0" ?>
<navtree xmlns="www.sl.com" version="1.0">
```

The `navtreedata` file must end with the following:

```
</navtree>
```

The following tags are supported:

| Tag | Description |
|---|---|
| `node` | Add a node to the navigation tree. |
| `treefont` | Set the font used in the navigation tree. |
| `treecolor` | Set font and background color in the navigation tree. Specify in hexadecimal RGB format: `#`*rrggbb* (for example, `#00FFFF` for cyan) or the following: `black`, `white`, `red`, `blue`, `green`, `yellow`, `cyan`, `magenta`, `gray`, `lightGray`, `darkGray`, `orange`, `pink`. |

The `node` tag supports the following attributes:

| Attribute | Description |
|---|---|
| `display` | Name of the display (`.rtv`) file. |
| `label` | Label for this node in the navigation tree. Defaults to display name if no label is set. Specify the font and color of the label using HTML. For example, to draw a green label using a 50-point italic monospaced font: |

| Attribute | Description |
|---|---|
| | `label="<html><p style='font-family:monospaced;font-style:italic;font-size:50;color:green'>Your Label Goes Here"`<br>HTML font settings specified here override `treecolor` and `treefont` settings for this node. |
| `mode` | If the attribute value is `keepalive`, the display is kept in memory the entire time the application is running. |
| `subs` | Substitutions to apply to the display. Substitutions are optional and must use the following syntax:<br>`$subname:subvalue $subname2:subvalue2`<br>If a substitution value contains a single quote, it must be escaped using a `/` :<br>`$filter:Plant=/'Dallas/'`<br>If a substitution value contains a space, it must be enclosed in single quotes. Do not escape these single quotes:<br>`$subname:subvalue $subname2:'sub value 2'`<br>A substitution string cannot contain any of the following characters:<br>`:` `\|` `.` tab space `,` `=` `<` `>` `'` `"` `&` `/` `\` `{` `}` `[` `]` `(` `)` |

The `treefont` tag supports the following attributes:

| Attribute | Description |
|---|---|
| `name` | Specifies the font family name. |
| `style` | Can be set to `plain`, `bold`, `italic`, or `bold italic`. |
| `size` | Font point size. |

The `treecolor` tag supports the following attributes:

| Attribute | Description |
|---|---|
| `text` | Specifies the font color for tree labels. |
| `background` | Specifies the background color for the tree and non-selected tree labels. |
| `selection` | Specifies the background color for a selected tree label. |

Here is an example:

```
<?xml version="1.0" ?>
<navtree xmlns="www.sl.com" version="1.0">
<node label="Nav Tree Example">
  <node label="Main Displays" display="main_panel">
    <node label="100 Displays">
      <node label="Panel 101" mode="keepalive" display="med_panel"
         subs="$title:101">
      </node>
      <node label="Panel 102" mode="keepalive" display="med_panel"
         subs="$title:102">
      </node>
      <node label="Panel 103" mode="keepalive" display="med_panel"
         subs="$title:103">
      </node>
    </node>
    <node label="200 Displays">
      <node label="Panel 201" display="med_panel" subs="$title:201">
      </node>
      <node label="Panel 202" display="med_panel" subs="$title:202">
      </node>
      <node label="Panel 203" display="med_panel" subs="$title:203">
      </node>
    </node>
  </node>
</node>
</navtree>
```

Nodes can be nested. You can only specify one top-level node.

#### *Using the RTViewNavTreePanel tag*

The tags described in this topic are deprecated. They will be removed in a future release. You should change to the new tags. See "Using new tags to configure the panels in a window" on page 189.

The `RTViewNavTreePanel` tag supports the following attributes:

| Attribute | Description |
|---|---|
| navtreedata | Name of the navigation tree definition file. This XML file must describe the elements of the tree. |
| lineStyle | Set the line style used in the navigation tree. Valid values are Angled and Horizontal. |
| region | Set the location of this RTViewNavTreePanel if it is in a BorderPanel. Valid values are west, east, center, north, and south. |
| height | Set the initial height of the RTViewNavTreePanel. |
| width | Set the initial width of the RTViewNavTreePanel. |

#### *Using the RTViewPanel tag*

The tags described in this topic are deprecated. They will be removed in a future release. You should change to the new tags. See "Using new tags to configure the panels in a window" on page 189.

The RTViewPanel tag supports the following attributes:

| Attribute | Description |
|---|---|
| display | Name of display (.rtv) file to load into the panel. |
| name | Corresponds to Window Name entered in the Drill Down Properties dialog. When using tabbed panels, if the name is not specified, a name is constructed internally using the display name and substitutions to make it easy to drill down between tabs. In this case, when you drill down from a tab using the Current Window option and the specified display with the specified substitutions is already loaded in another tab, the Dashboard Viewer will switch to that tab. |

| Attribute | Description |
|---|---|
| region | Set the location of this RTViewPanel if it is in a BorderPanel. Valid values are west, east, center, north, and south. |
| scrollbars | Control the visibility of scroll bars in the panel. The permitted values are as-needed, never, and always. The default value is as-needed. In some cases, setting the scrollbars attribute to never on title or footer panels can improve the resize behavior of the Dashboard Viewer. |
| subs | Specify initial substitutions for this panel. Substitutions are optional and must use the following syntax:<br>`$subname:subvalue $subname2:subvalue2`<br>If a substitution value contains a single quote, it must be escaped using a / :<br>`$filter:Plant=/'Dallas/'`<br>If a substitution value contains a space, it must be enclosed in single quotes. Do not escape these single quotes:<br>`$subname:subvalue $subname2:'sub value 2'`<br>A substitution string cannot contain any of the following characters:<br>: \| . tab space , = < > ' " & / \ { } [ ] ( )<br>**Note** Substitutions set in **Application Options** will apply to all displays. |
| title | Set the title of the tab containing this RTViewPanel. This is only used if the RTViewPanel is in a TabbedPanel. |

# 7 Sending Events to Correlators


- Using the Define Apama Command dialog ..... 226
- Send event authorization ..... 232

Dashboard Builder supports the creation of Dashboard commands that send user-defined events, just as it supports the creation of commands that send predefined Scenario manipulation commands.

## Using the Define Apama Command dialog

As with Scenario-management commands, you define a send-event command by associating it with the `command`, `actionCommand`, or `commandString` property of a Dashboard object such as a button.

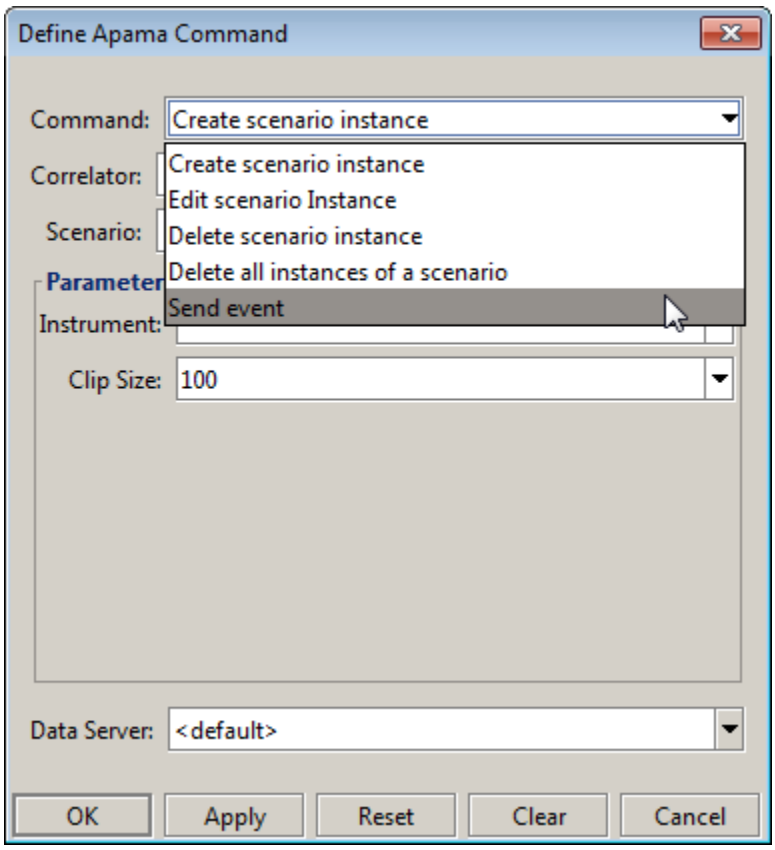
**To make the association**

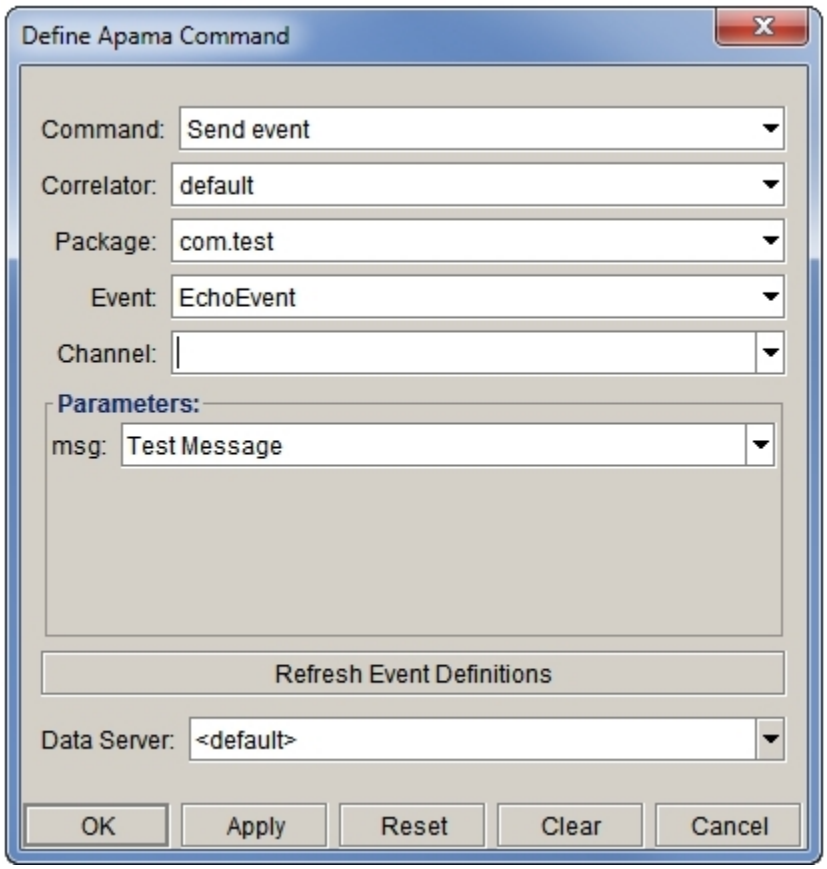
1. Select the Dashboard object.
2. Right-click the `actionCommand` or `commandString` property in the Object Properties panel.
3. Select **Define Command > Apama** from the popup menu.

The **Define Apama Command** dialog appears.

### Command field

The choices for the **Command** field include **Send event**.






## Package field

The choices for the **Package** field include all the packages for the selected correlator. Events that do not have a package are grouped under the package "default".


## Event field

The choices for the **Event** field include all the events for the selected package.

## Channel field

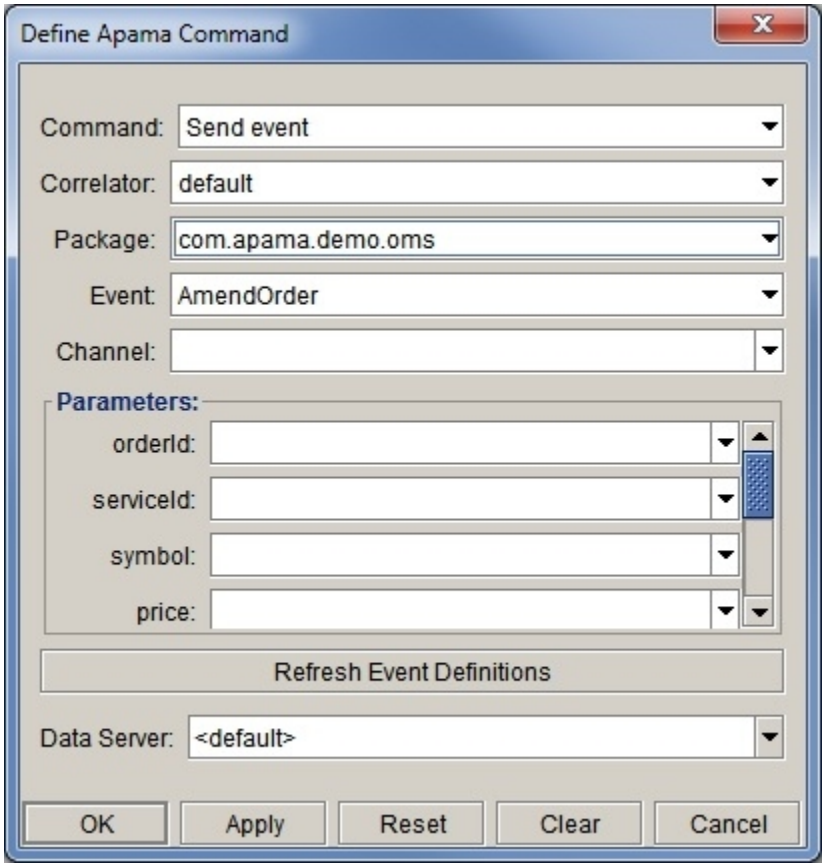
Optionally, specify a channel on which to send the event. For example:

```
orders
```

If you do not specify a channel then the default channel is used.

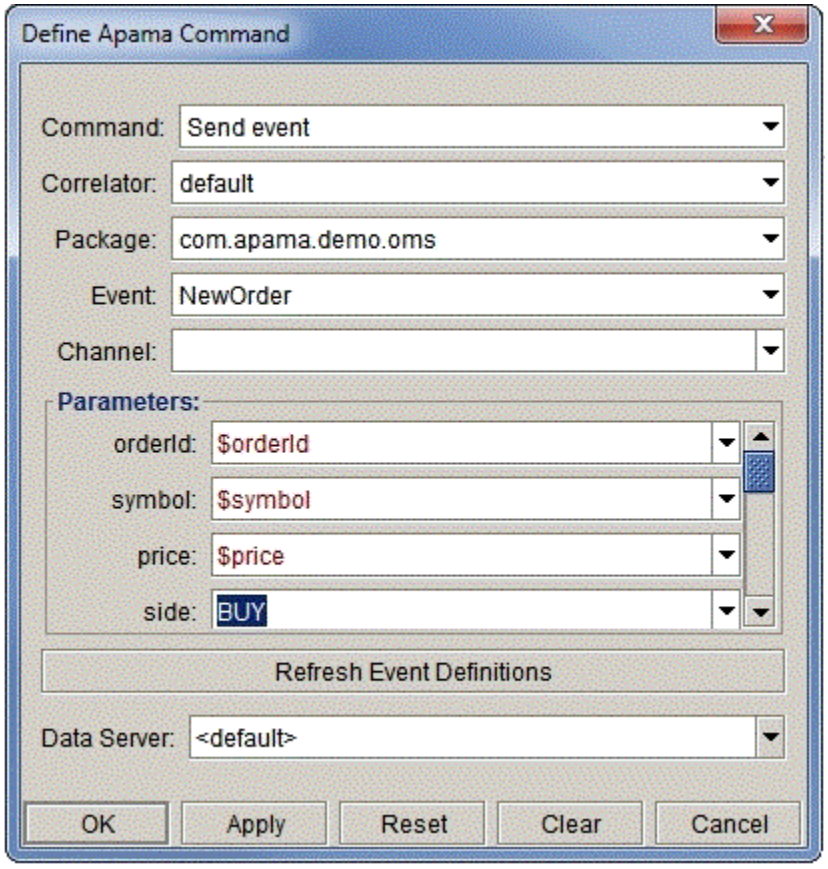
## Other dialog fields

The remaining fields shown are dependent on the event selected; one field is shown in the dialog for each field in the event.




As with other commands, the value of each event field can be attached to a Dashboard variable or set to a hard coded value.

In the following example `orderId`, `symbol`, `price`, and `quantity` are attached to dashboard variables, `side` and `type` (you would scroll down to see these fields) are fixed, and other fields (again, scroll down to see them) are not set:




***Default values***

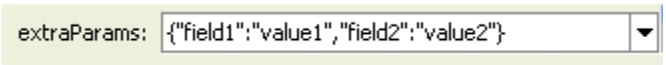
It is not necessary to set each event field in order to send an event. Empty fields are set to default values depending on type:.

| Type | Default Value |
|---|---|
| `boolean` | `false` |
| `integer` | `0` |
| `float` | `0.0` |
| `string` | `""` |
| `location` | `(0.0,0.0,0.0,0.0)` |
| `sequence` | `[]` |
| `dictionary` | `{}` |

| Type | Default Value |
|---|---|
| `event` | *`event_name (default fields)`* |

#### *Specifying values for complex types*

You specify values for complex types (`location`, `sequence`, `dictionary`, or `event`) by using the format specified in the *EPL Reference*. For example to specify **extraParams** you could specify the value as follows:




You can also use dashboard variables, or a single dashboard variable that contains the entire value, for example:


where `$extraParams` equals `{"field1":"value1","field2":"value2"}`.

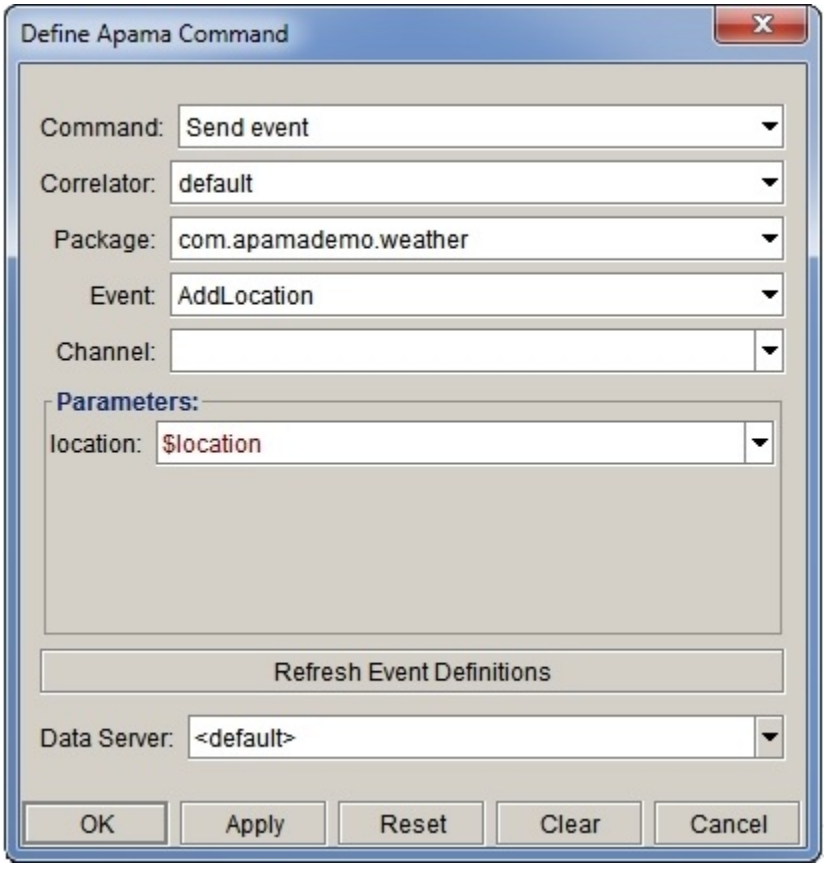
## Updating event definitions in Builder

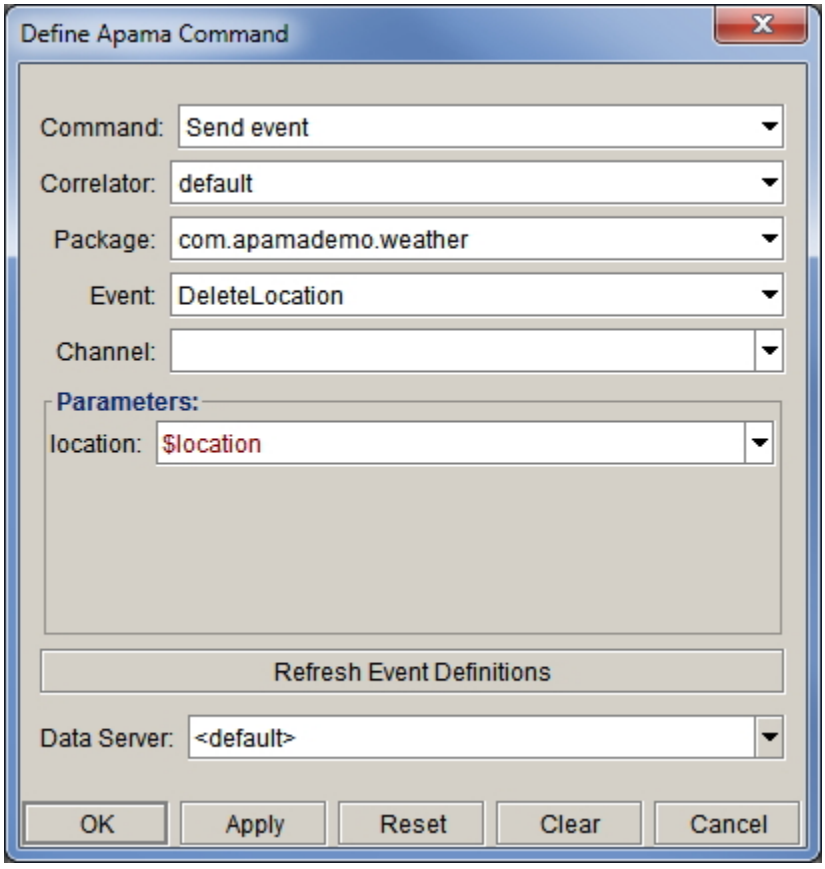
The Dashboard Builder retrieves the latest event definitions from each correlator at startup. If the event definitions change, you can force them to be refreshed by using the **Refresh Event Definitions** button at the bottom of the Define Apama Command dialog.

Note that the **Refresh Event Definition** button only updates the event definitions for the selected correlator.

## Example

The **Weather** demo which can be accessed via Software AG Designer's Welcome page (go to **Help > Welcome**, and then click **Demos** under the **Apama** heading) uses the following dialogs to define actions for the **Add Location** and **Delete Location** buttons:


Define Apama Command
Command:
Send event
Correlator:
default
Package:
com.apamademo.weather
Event:
AddLocation
Channel:
Parameters:
location:
$location
Refresh Event Definitions
Data Server:
<default>
OK
Apply
Reset
Clear
Cancel




## Send event authorization

By default, any user is authorized to send any event. However you can create a custom *event authority* that determines whether a given user is authorized to send a given event. An event authority is a Java class that implements the `canSend` method of the interface `com.apama.dashboard.security.IEventAuthority`:

```
boolean canSend (IUserCredentials credentials, Event event);
```

If `canSend()` returns `true` the user is allowed to send the event. If it returns `false` the user is not allowed to send the event and the attempt to send the event is treated as a command failure. Dashboard object property settings determine if this error is displayed to the user.

The event authority is specified in the `EXTENSIONS.ini` file in the `lib` directory of your Apama installation. Here is a portion of `EXTENSIONS.ini` as shipped:

```
# List of event authorities. An event authority is called to determine
# if a user has rights to send an event to a correlator. Each must implement
# the interface:
##   com.apama.dashboard.security.IEventAuthority
## Multiple authorities can be specified. They will be called in the order
# listed.
## Format:
##  eventAuthority <classname>
## NoOpEventAuthority - Allows all users to send events
```

```
eventAuthority com.apama.dashboard.security.NoOpEventAuthority
# DenyAllEventAuthority - Allows no users to send events
#eventAuthority com.apama.dashboard.security.DenyAllEventAuthority
# eventAuthority <your_class_name_here>
```

Two event authorities are provided with your installation:

- `com.apama.dashboard.security.NoOpEventAuthority`: Permits all users to send any event.
- `com.apama.dashboard.security.DenyAllEventAuthority`: Denies all users rights to send any event.

`NoOpEventAuthority` is the default event authority. Use a custom event authority when deploying your Dashboards.

See *Deploying and Managing Apama* for more information on customizing authorization.

# 8 Using XML Data


- XML data format ..... 236
- Defining an XML data source ..... 238
- Attaching objects to XML data ..... 239

Dashboard Builder enables you to augment your dashboard by using XML data files as a data source in addition to Apama scenarios and DataViews. The properties of dashboard objects can be attached to data elements in XML files. To be used as a data source, an XML file must follow the formatting guidelines presented in this chapter.

XML files can be used to make a dashboard more generic by isolating label values, colors, and similar attributes in a file which can be shared by multiple dashboards. XML files can also be used as an intermediary for bringing data from other sources into a dashboard.

## XML data format

XML files used as data sources with Dashboard Builder must adhere to the formatting guidelines detailed in this section.

XML data files must contain the `dataset` element. This element identifies the XML data as a dashboard XML data file. The standard template for an XML data file is as follows:

```
<?xml version="1.0"?>
<dataset xmlns="www.sl.com" version="1.0">Data elements</dataset>
```

All XML data files must adhere to this template.

XML data files can contain both scalar and tabular data elements as discussed in the topics below. XML data files can contain multiple scalar and tabular data elements.

### Scalar data elements

Scalar data elements are single values such as a string or number. Scalar data elements are useful for isolating common labels, colors, and similar items in XML resource files that can be shared by multiple dashboard files.

A scalar element is defined in an XML data file with the `data` tag as follows:

```
<data key="element_name" value="element_value" />
```

The `key` attribute specifies the name of the data element. This name will be used when attaching object properties to the data element. The value specifies the value for the element; both string and number values can be specified for the value.

Following is an example of an XML data file containing scalar data elements:

```
<?xml version="1.0"?>
<dataset xmlns="www.sl.com" version="1.0">
  <data key="status_label" value="Current Status:" />
  <data key="status_complete" value="Completed" />
  <data key="status_failed" value="Failed" />
  <data key="load_factor" value="1.5" />
  <data key="max_occurence" value="10000" />
</dataset>
```

Here, five different scalar data elements are defined. The first three, `status_label`, `status_complete`, and `status_failed`, have string values. The last two, `load_factor` and `max_occurence`, have number values.

## Tabular data elements

Tabular data elements contain multiple columns and rows of data. The value for each field can be a string, integer, double, or boolean. Tabular data elements are useful for data sets containing multiple item instances. Tabular data can be used to populate Table, Trend Chart, and other dashboard objects.

Tabular elements are defined in a `table` tag that includes a set of tags that describe the data in the table and tags for each row of data values. A tabular element is defined as follows:

```
<table key="production_table">
  <tc name="column1"
    type="string | double | int | boolean"
    index="true | false"/>
  <tc name="column2"
    type="string | double | int | boolean"
    index="true | false"/>
  <tr name="ID0”>
    <td>column1_value</td>
    <td>column2_value</td>
  </tr>
  <tr name="ID1">
    <td>column1_value</td>
    <td>column2_value</td>
  </tr>
</table>
```

The `key` attribute on the table tag specifies the name of the data table. This name will be used when attaching object properties to the data element.

The `tc` tag defines a column in the table. For each column, you must specify a name, type, and whether or not the column is to be used as index. Subsequent row definitions must contain values for each column where the type of the value matches the type defined for the column. The `index` field is reserved for future use.

The `tr` tag defines a single row of data. Each row must contain a `td` tag for each column in the table. The `td` tags define the value for a column for a single row.

Following is an example XML data file containing a tabular data element named `production_table`:

```
<?xml version="1.0"?>
<dataset xmlns="www.sl.com" version="1.0">
<table key="production_table">
  <tc name="Plant"
     type="string"
     index="true"/>
  <tc name="Units in Production"
     type="int"
     index="false"/>
  <tc name="Units Completed"
     type="int"
     index="false"/>
  <tc name="Status"
     type="string"
     index="false"/>
  <tc name="On Schedule"
     type="boolean"
```

```
    index="false"/>
  <tr name="PID 0">
    <td>San Francisco</td>
    <td>87</td>
    <td>70</td>
    <td>online</td>
    <td>true</td>
  </tr>
  <tr name="PID 1">
    <td>San Jose</td>
    <td>75</td>
    <td>63</td>
    <td>online</td>
    <td>false</td>
  </tr>
</table>
</dataset>
```

Here, the table is defined as containing four columns; `Plant`, `Units in Production`, `Units Completed`, and `On Schedule`. There are two rows in the table; one each for San Francisco and San Jose.

## Defining an XML data source

**To attach object properties to data elements in an XML data file, you need to first make the XML data file known by adding it as a data source**

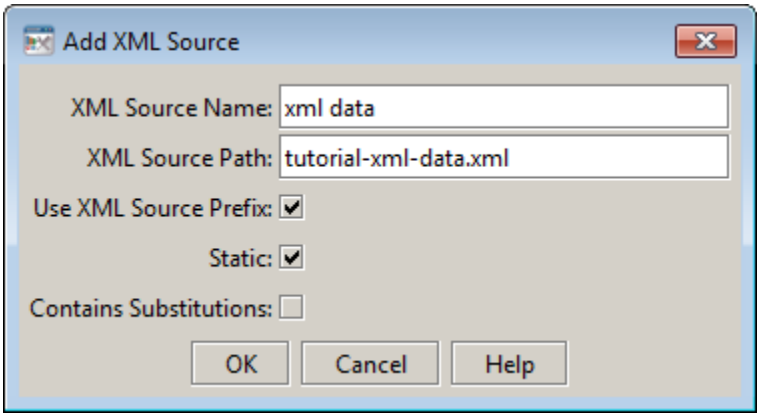
1. Select **Options...** from the **Tools** menu. This will display the Application Options dialog.
2. In the Application Options dialog select **XML** in the left pane.

   On this tab you can define the XML files to be used as data sources. The **XML Source Prefix** field allows you to define a file path prefix that can be used to locate XML data files.
3. Set the **XML Source Prefix** field to the directory of the tutorial sample in your Apama installation. By default this is:

   ```
   %APAMA_HOME%\samples\dashboard_studio\tutorial\
   ```

   Be sure to include the final backslash in the **XML Source Prefix**.
4. Click the **Add** button to define a new XML data source.

   This will display the Edit XML Source dialog.
5. Define a new data source as follows and click the **OK** button.




You have defined the XML data source named `xml data`. The data for this data source is in the file `tutorial-xml-data.xml` located in the tutorial directory.

When defining an XML data source you specify:

- **XML Source Name** — The name you will use to refer to the data source when defining data attachments.
- **XML Source Path** — The full path to the XML data file. If an XML source prefix is used, a partial path only need be specified.
- **Use XML Source Prefix** — If enabled, the XML source prefix will be affixed to the XML source path.
- **Static** — If enabled Apama will read the file only once. If disabled, Apama will read the file each time it is modified. Each time the file is read any attached objects will update to show the latest data elements in the file.
- **Contains Substitutions** — Enable this option if the XML source path contains substitution variables. If enabled, Apama will not read the file until the substitutions have been defined.

To edit an existing XML data source, double-click it in the list of XML sources. You can also specify an XML data source to use as the default when defining XML data attachments.

XML data source definitions are saved in `OPTIONS.ini`. To persist an XML data source definition you must click **Save** in the Application Options dialog.

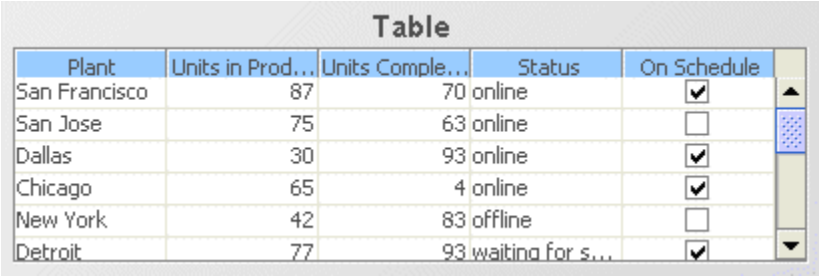
## Attaching objects to XML data

After having defined an XML data source you can attach object properties to the data elements within the XML data file. The steps for doing this are similar to defining attachments to scenario data.

**To attached object properties to the data elements within the XML data file**

1. If you have not yet done so, define the XML data source `xml data` as detailed in the previous section.

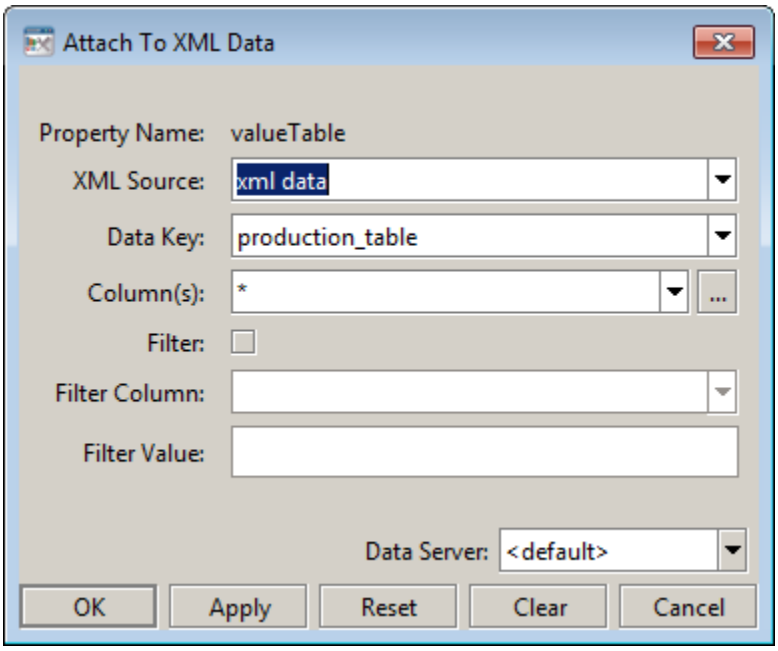
2. Open the file `tutorial-xml-data.rtv` by selecting **XML Data** on the tutorial main page.




The table object in this dashboard is attached to the `production_table` data element in the file `tutorial-xml-data.xml`.

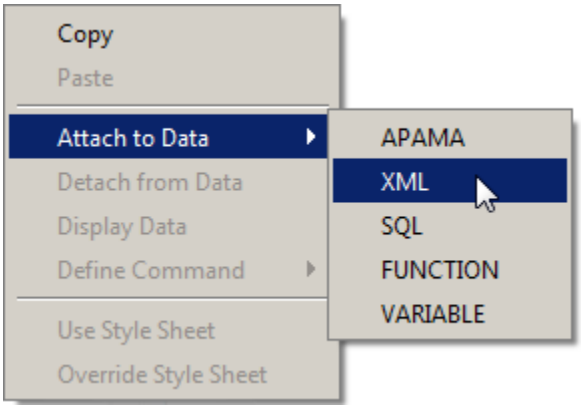
3. Open the file `tutorial-xml-data.xml` in a text editor and examine it to see that there is a column in the table for each column defined for `production_table` and that there is a row in the table for each row defined.

4. Select the table object and double-click the `valueTable` property in the Object Properties panel.

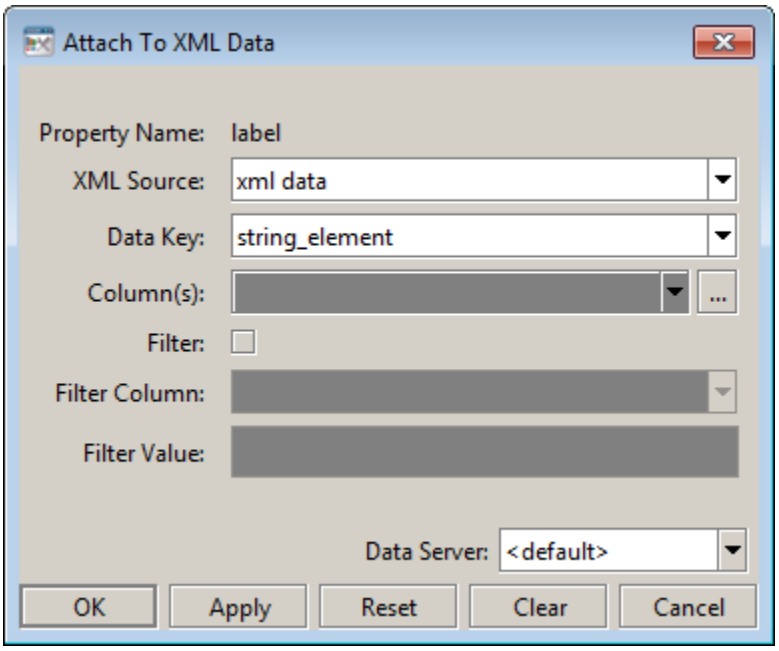



Here the property is attached to the `production_table` data element for the XML data source named `xml data`. The `Columns` and `Filter` fields can be used to select a subset of columns or rows as is done for scenario data attachments.

5. With the table object still selected, right-click the `label` property and select **XML** from the **Attach to Data** menu.

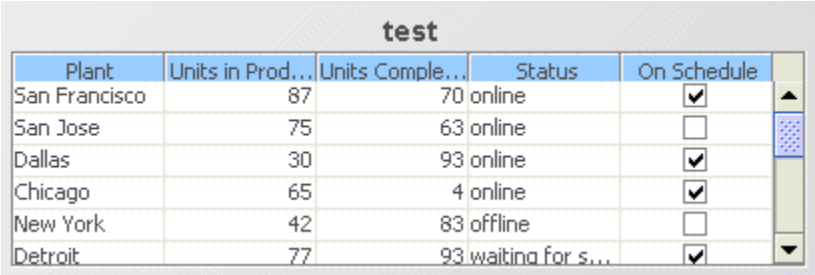



6. If `label` is a scalar property, it must be attached to a scalar data element. Attach it to the data element `string_element` as shown in the following:




Do not use the **Data Server** field of the Attach to XML Data dialog.

The label of the table will change.



# 9 Using SQL Data


- SQL system requirements and setup ..... 244
- Attaching visualization objects to SQL data ..... 244
- Defining SQL commands ..... 249
- Specifying application options ..... 251
- Deploying applet and WebStart dashboards ..... 255
- Setting up SQL database connections ..... 255
- Setting SQL data source options ..... 257

The SQL data source provides access to JDBC enabled databases. The Attach to SQL Data dialog makes it easy to browse, select data tables, filter information, and institute query policies with a simple user interface. For those familiar with SQL, it is also possible to enter SQL commands to specify database queries.

## SQL system requirements and setup

The SQL data source requires a database with a JDBC driver. In addition, if you use the applet deployment, you will need to set up applet permissions on each client to allow access to your database. See "Setting up SQL database connections" on page 255.

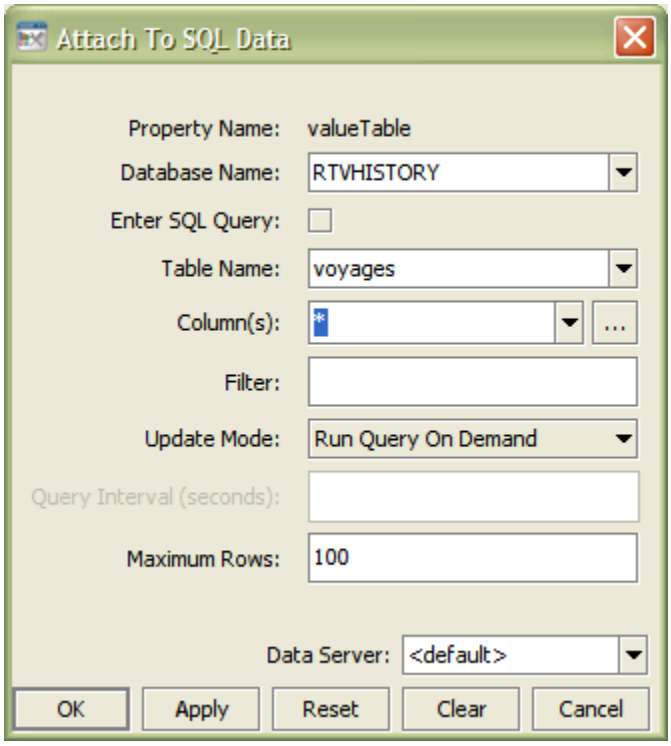
You must also modify your **Dashboard Properties** (select **Properties** from the **Project** menu in Software AG Designer). In order to use a direct JDBC connection to communicate with a database, add your JDBC `jar` file to your **Dashboard Properties**.

## Attaching visualization objects to SQL data

From the Object Properties window you can access the Attach to SQL Data dialog, which is used to connect an object to your database using an SQL query. Once an object has been attached to your database it can receive periodic or on-demand updates.

When an object property is attached to data, the **Property Name** and **Value** in the **Object Properties** window will be displayed in green. This indicates that editing this value from the **Object Properties** window is no longer possible.

To remove the data attachment and resume editing capabilities in the Object Properties window, right-click the **Property Name** and select **Detach from Data**. You will recognize that an object property has been detached from the database when the **Property Name** and **Value** are no longer green.




Use the `--sql quote` command line option to enclose all table and column names specified in the Attach to SQL Data dialog in quotes when an SQL query is run. This is useful when attaching to databases that support quoted case-sensitive table and column names. *Note:* If a case-sensitive table or column name is used in the **Filter** field, or you are entering an advanced query in the **SQL Query** field, they must be entered in quotes even if the `--sql quote` option is specified.

**To connect an object to your database using an SQL query**

1. Right-click the **Property Name** from the Object Properties window and select **Attach to Data** > **SQL**.

   The Attach to SQL Data dialog displays.

   The Attach to SQL Data dialog provides drop down menus and an optional filter field that allow you to specify information that will be used to create an SQL query for the selected database. Alternatively, select the **Enter SQL Query** checkbox in order to enter an advanced query.

2. From the **Database Name** drop down menu, select the name of database to query.

   The **Database Name** drop down menu lists all available databases. The **Database Name** field automatically displays the name of the default database. If the item you require is not listed, type your selection into the field.

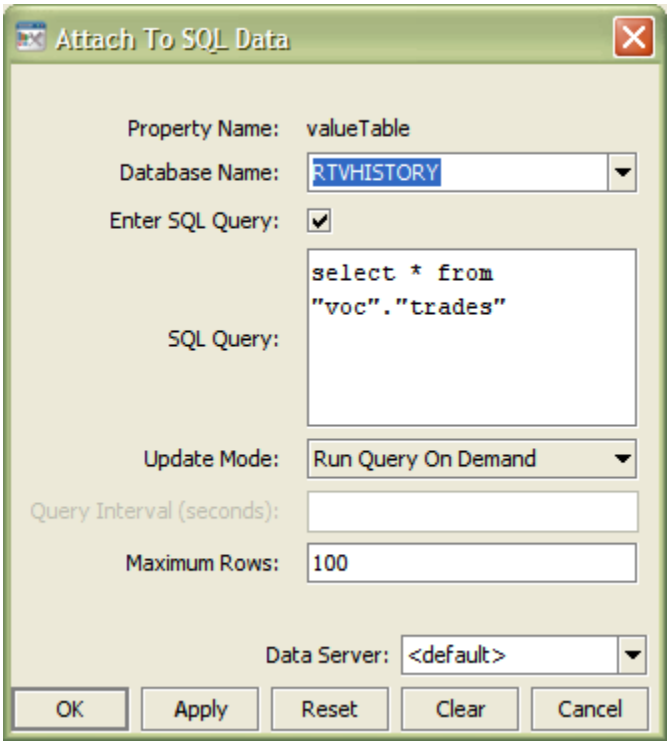
   A Database Repository file can be used to populate the initial values of drop down menus for **Table Name** and **Column(s)**. See "Specifying application options" on page 251 for information on how to create a Database Repository file. Otherwise, drop

down menus populate based on databases added from the Application Options dialog or those typed directly into the **Database Name** field.

3. Check the **Enter SQL Query** checkbox in order to enter an advanced query.

   If selected, the **SQL Query** text field, where you can enter your query, will replace the **Table Name**, **Column(s)** and **Filter** fields.

   **Note:** This option is for advanced users; SQL syntax will not be validated or checked for errors




4. In the **Table Name** field, enter the name of table in database to query.

   You can create a file to exclude tables from the **Table Name** drop down menu. See "Setting up SQL database connections" on page 255 for details.

5. From the **Column(s)** pull down menu, select the columns in table to display.

   A Database Repository file can be used to populate the initial values of drop down menus for **Table Name** and **Column(s)**. See "Specifying application options" on page 251 for information on how to create a Database Repository file.

6. In the **Filter** field, optionally, enter SQL filter to apply to query.

   Uses standard SQL syntax.

7. From the **Update Mode** pull down menu, select one of the following:

   - **Run Query Once:** Select this if the data returned by this query is static. If selected, Apama will run this query only once. This is the default setting.

- **Run Query Every Update Period:** Select to run this query each update period. See "Specifying application options" on page 251 for information on setting the update period.
- **Run Query Every Query Interval:** Select to run this query once every Query Interval.
- **Run Query On Demand:** Select to run this query each time a display that uses the query is opened and each time a substitution string that appears in the query string has changed.

8. In the **Query Interval (seconds)** field, enter the time in seconds to control how often Apama will run this query.

   **Note:** The query interval is evaluated during each update pass, so the amount of time elapsed between queries may be longer than the value entered. For example, if the update period is 2 seconds and the query interval is 5 seconds, the query will get run every six seconds. This option is only available if the **Update Mode** is **Run Query Every Query Interval**.

9. In the **Maximum Rows** field, enter the maximum number of rows to return from this query.

   **Note:** On some objects an additional property may further reduce the number of data points displayed. For example, the `maxNumberOfRows` property on the table or the `maxPointsPerTrace` property on the trend graph.

10. Do not modify **Data Server** field.
11. Click **OK** to apply the value and close the dialog.

    You can also choose the following:

    - **Apply:** Applies values without closing the dialog.
    - **Reset:** Resets all fields to last values applied.
    - **Clear:** Clears all fields. Detaches object from database (once **Apply** or **OK** is selected).
    - **Cancel:** Closes the dialog with last values applied.

    **Note:** By default Apama will attempt to communicate with your database using a connection that is not password protected. If you are using a direct JDBC connection you will need to add your database in **Application Options | SQL**.

## Validation colors

Fields in the dialog change colors according to the information entered. These colors indicate whether or not information is valid. Information entered into the dialog is validated against the selected database or the Database Repository file. See "Specifying application options" on page 251 for information on how to create a Database Repository file. *Note:* Filters and advanced SQL queries are not validated.

The following describes the significance of the **Attach to SQL Data** validation colors:

- Blue: Unknown, that is, entry does not match any known database (or you have not attempted a connection—see *Note* below).
- Yellow: Offline, that is, not connected to database.
- White: Valid.
- Red: Invalid. **Database** is valid, but **Table** or **Column(s)** selected are not.

**Note:** If a database is unknown, when you click **OK** or **Apply** Apama will attempt to communicate with it using using the defined connection. If the validation response remains unknown, see "SQL tab" on page 251 for information on how to add a database. If you are using a direct JDBC connection you will need to add your database in **Application Options**.

## Substitutions

Substitutions allow you to build open-ended displays in which data attachments depend on values defined at the time the display is run. Generic names, such as `$table1` and `$table2`, are used instead of specific values. Later when the display is running, these generic values are defined by the actual names, such as `production_table` and `system_table`. In this way, a single display can be reused to show data from a number of different databases.

## Select table columns

From the Attach to SQL Data dialog you can specify which table columns to display and in what order they will appear. In order to populate tthe listing of available columns, you must first select a valid database and table.

**To specify which table columns to display and in what order they will appear**

1. Right-click the **Property Name** from the Object Properties window and select **Attach to Data** > **SQL**.

   The Attach to SQL Data dialog displays.

2. Click on the ellipses button in the **Column(s)** field (or right-click in the **Column(s)** field and click **Select Columns**).

   The **Select Columns** dialog displays, which contains a list of **Available Columns** that you can add to your table.

3. To add a column, select an item from the **Available Columns** list and click **Add** button.

   If the item you require is not listed, type your selection into the **Enter Column Name** field.

4. Click the **Remove** button to delete an item previously added to the **Selected Columns** list.

5. Control the order of columns in a table by arranging the items in the **Selected Columns** list with the **Move Up** and **Move Down** buttons.

Validation colors indicate whether selected columns are valid. However, if even one column selected is invalid the **Column(s)** field in the **Attach to SQL Data** dialog will register as an invalid entry.

**Note:** Invalid columns will not update.

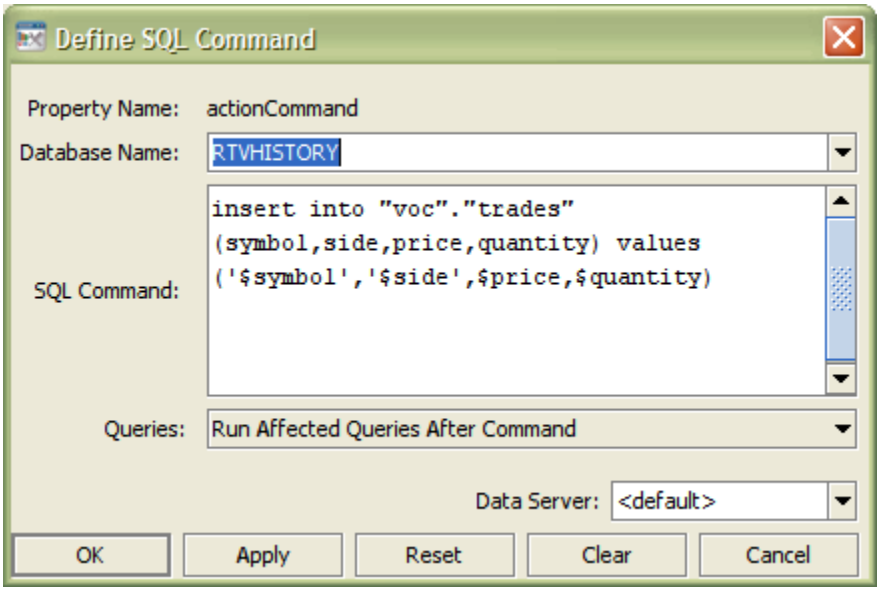
## Defining SQL commands

From the Object Properties window you can access the Define SQL Command dialog. This dialog is used to assign SQL commands allowing you to issue commands from within a dashboard.

### To assign SQL commands

1. Right-click the appropriate command property in the Object Properties window and select **Define Command** > **SQL**.

   The Define SQL Command dialog displays, which provides a drop down menu with available databases and a field to enter a SQL statement.




2. In the **Database Name** drop down menu, enter the name of database to query.

   The **Database Name** drop down menu lists all available databases. The **Database Name** field automatically displays the name of the default database. If the item you require is not listed, type your selection into the field. Drop down menus populate based on databases added from the Application Options dialog or those typed directly into the **Database Name** field.

3. In the **SQL Command** field, enter a SQL statement to execute using standard SQL syntax.

   **Note:** This option is for advanced users, SQL syntax will not be validated or checked for errors.

4. In the **Queries** field, if **Run Affected Queries After Command** is selected, Apama immediately runs all queries, including static queries, that use the database table modified by the command. This causes table changes to be displayed immediately, rather than waiting for the next scheduled query update.

   This option is only supported for update, insert, and delete operations in which the name of the database table to be modified is specified explicitly. If a command performs another SQL operation (such as running a stored procedure that modifies tables), the results of the operation will not be displayed until the next scheduled update of each affected query. Display of the modified data may be delayed for other reasons, for example, if the database does not commit the results immediately and instead returns the old data on the next query.

5. Do not modify the **Data Server** field.

6. Click **OK** to apply the value and close the dialog.

   You can also choose the following:

   - **Apply:** Applies values without closing the dialog.
   - **Reset:** Resets all fields to last values applied.
   - **Clear:** Clears all fields. Detaches object from assigned command (once **Apply** or **OK** is selected).
   - **Cancel:** Closes the dialog with last values applied.

## Validation colors

The **Database Name** field changes colors according to the information entered. These colors indicate whether or not information is valid. Information entered into the dialog is validated against the selected database or the Database Repository file. See "Specifying application options" on page 251 for information on how to create a Database Repository file.

**Note:** The SQL Command field is not validated.

The following describes the significance of the Define SQL Command validation colors:

- Blue: Unknown. Entry does not match any known database (or you have not attempted a connection—See *Note* below).
- Yellow: Offline. Not connected to specified database.
- White: Valid. Database name is valid.

**Note:** If a database is unknown, when you click **OK** or **Apply** Apama will attempt to communicate with it using the defined connection. If the validation response remains unknown, see "SQL tab" on page 251 for information on how to add a database. If you are using a direct JDBC connection, you will need to add your database in **Application Options**.

### Special values

When an actionCommand is executed $value is replaced with the value from the control. This value may be used in any field in the Define SQL Command dialog.

**Note:** This value may only be used for Action Commands.

## Specifying application options

To access the Application Options dialog, in the Builder select **Tools** > **Options**.

Options specified in the SQL tab can be saved in an initialization file (`OPTIONS.ini`). On startup, the initialization file is read by the Builder, Viewer, Display Server, and Data Server to set initial values. If no directory has been specified for your initialization files and `OPTIONS.ini` is not found in the directory where you started the application, then Apama will search under `lib` in your installation directory.

**Note:** Options specified using command line arguments will override values set in initialization files.

### SQL tab

This tab allows you to add or remove your databases and set the default database. In order for Apama to communicate with your databases, you must set up a JDBC connection.

When you add a database to the list it will be highlighted in yellow indicating that it is not connected. To attempt to connect to a database, click **OK**, **Apply**, or **Save**. If the background remains yellow, then Apama was unable to make a connection to your database. *Note:* Databases that have been set up to **Use Client Credentials** will not connect unless you are logged in and you have objects in your display that are using that connection.

Check your database connection and see "Setting up SQL database connections" on page 255 for information on how to set up your driver correctly.

If the connection is successful, and the **Get Tables and Columns from Database** checkbox is selected, Apama will use information from this database to populate drop down menus in the Attach to Data dialog with available tables and columns. If a database repository is found, information from your database will be merged with data from the repository file. If you deselect the Get **Tables and Columns from Database** checkbox Apama will no longer query your database for this information, but the database repository will still be

used to populate drop down menus. Using a database repository to populate drop down menus makes it possible to specify which tables and columns from your database will be listed in the Attach to Data dialog and gives you the ability to build displays while databases are offline.

If you are using a direct JDBC connection you must click **Save** in order to record your options in `OPTIONS.ini`. This will allow Apama to reconnect with your database the next time you run the Builder or the Viewer.

**Note:** Regardless of which tab you are currently working from in the Application Options dialog, each time you click **OK**, **Apply**, or **Save** Apama will attempt to connect to all unconnected databases, except those that have **Use Client Credentials** checked.

The Application Options dialog has the following fields and buttons:

- **Default Database:** Name of database used as the default for data attachments. Select from drop down menu to change default setting.
- **Add Database:** Click to open the Add Database dialog. To edit, select a database from the list and double-click. Databases that are updating objects in a current display cannot be renamed.

## Adding a Database

The Add Database dialog has the following fields:

- **Database Name:** The name to use when referencing this database connection in your data attachments.
- **User Name:** The user name to pass into this database when making a connection. This parameter is optional.
- **Password:** The password to pass into this database when making a connection. This parameter is optional.
- **Use Client Credentials:** If selected, the user name and password from the Apama login will be used instead of the **User Name** and **Password** entered in the **Add Database** dialog. Connections to this database will only be made when you are running with login enabled and a display is opened that accesses this database.
- As a result, this connection will not be made when you click **OK** or **Apply** in the Application Options dialog and will remain yellow. If you will be using the Data Server or the Display Server with a database connection that has this option enabled, you must enable **Use Client Credentials for Database Login** in these applications.
- **Table Types:** Specify the types of tables to retrieve when querying the database for available tables. Refer to your database manual for a list of valid table types. This parameter is optional. Table types are entered as a comma delimited list, for example, `TABLE, VIEW`.
- **Run Queries Concurrently**: If selected, each query on the connection is run on its own execution thread. The default is disabled. *Note:* This option should be used with

caution since it may cause SQL errors when used with some database configurations and may degrade performance due to additional database server overhead. See your database documentation to see whether it supports concurrent queries on multiple threads.

- **JDBC Driver Class Name:** The fully qualified name of the JDBC driver class to use when connecting to this database. The path to this driver must be included in the `RTV_USERPATH` environment variable.
- **JDBC Database URL:** The full database URL to use when connecting to this database using the specified JDBC driver. Consult your JDBC driver documentation if you do not know the database URL syntax for your driver.
- **Remove Database:** Select a database from the list and click **Remove Database** to delete. Databases that are updating objects in a current display cannot be removed.
- **Suppress Permission Errors From Database:** If selected, SQL errors with the word "`permission`" in them will not be printed to the console. This is helpful if you have selected the **Use Client Credentials** option for a database. In this case, your login does not allow access for some data in their display, you will not see any errors.
- **Get Tables and Columns from Database:** If selected, information from your database will automatically populate drop down menus in the Attach to Data dialog and you will be able to select from available tables and columns in your database. *Note:* If a database repository is found, information from your database will be merged with data from the repository file.
- **Save Database Repository:** Click to save a file that records available tables and columns in your database and applies values to drop down menus in the Attach to Data dialog.

Instead of using the **Add Database** dialog, it is possible enter this information manually into `OPTIONS.ini`. See "Entering database information directory into OPTIONS.ini" on page 254.

## Database repository

Click **Save Database Repository** to save a file that contains available information for tables and columns in your database. Before saving a database repository, you must add the database or databases from which the file will retain information.

> **Note:** If Apama does not make a connection with your database, then information from that database cannot be saved to the database repository file.

Information stored in the database repository file will be used to populate the initial values of drop down menus in the Attach to Data dialog. *Note:* The saved file will be named `sqlrepository.xml`. If the name of the database repository file is changed, Apama will not be able to locate the file. As a result, drop down menus will populate based on databases added from the Application Options dialog or those typed directly into the Attach to Data dialog.

When you click **Save Database Repository**, a confirmation dialog will appear to verify in which directory you would like to save the database repository file. If you specified a directory for your initialization files, all repository files will be saved to, and read from, that directory. If you select the `lib` directory, the repository file will be available from any directory where you run Apama. If you do not select the `lib` directory, the repository file will be saved in the directory where you started the current session and will only be available when you run Apama from that particular directory.

See "Setting up SQL database connections" on page 255 for details on editing an existing database repository file.

## Entering database information directory into OPTIONS.ini

To add an SQL database by entering information directly into `OPTIONS.ini` (instead of using the **Add Database** dialog—see "Adding a Database" on page 252), add a line of text of the following form:

```
sqldb databaseName username password jdbcUrl jdbcClassName tableTypes
   useClientCredentials-boolean runQueriesConcurrently-boolean
```

You must supply all fields. Use "**-**" for fields that do not have a value.

Following is an example:

```
sqldb myDatabase - - - - - false false
```

In the example above, the databaseName is `myDatabase`, and both useClientCredentials and runQueriesConcurrently are `false`. All other fields are not specified.

For JDBC databases `jdbcUrl` and `jdbcClassName` must be set.

See also "Generating encrypted passwords for SQL data sources" on page 254.

### *Generating encrypted passwords for SQL data sources*

If you are adding an SQL data source by entering information directly into `OPTIONS.ini` (see "Entering database information directory into OPTIONS.ini" on page 254), and you specify a username and password, use the `dashboard_management` utility in order to generate an encrypted version of the password. Use the encrypted version in the **sqldb** line of `OPTIONS.ini`.

Commands of the following form yield the encrypted string as output:

```
dashboard_management -e | --encryptString password
```

Following is an example:

```
dashboard_management -e sunshine
```

This yields the following output:

```
013490135101344013490133801339013440133 5
```

Following is a sample **sqldb** line that includes the encrypted password shown above:

```
sqldb test2 username 013490135101344013490133801339013440133 5 - - - true false
```

## Deploying applet and WebStart dashboards

This page contains details about the deployment process that are specific to the SQL data source. See *Deploying Apama Applications* for general information about dashboard deployment.

If you will be using applet or WebStart dashboards that include SQL data attachments, modify your Java security settings to include the following permission:

```
permission java.util.PropertyPermission "file.encoding", "read";
```

If you will be accessing a database on another system, modify your Java security settings to include the following permission:

```
permission java.net.SocketPermission "host", "accept, connect, listen, resolve";
```

Where *host* is the system where the database is running.

If you are using a JDBC driver to connect to your database, include the `jar` for your driver in the `ARCHIVE` parameter. Depending on your driver, you may also need to add an `accessClassInPackage` RuntimePermission for your driver package.

## Setting up SQL database connections

Apama communicates with your database using a direct JDBC connection that requires some set up before Apama can communicate with your database.

Once you have set up your database connection, you will need to add your database in the Builder from the Application Options dialog on the SQL tab (see "SQL tab" on page 251). Apama will attempt to connect to your database. If Apama is unable to connect to your database, this means that either the driver is not set up correctly or that you do not have permission to access the database. *Note:* Databases that have been set up to **Use Client Credentials** will not connect unless you are logged in and you have objects in your display that are using that connection.

If the connection is successful, and the **Get Tables and Columns from Database** checkbox is selected in the Application Options dialog, Apama will use information from this database to populate drop down menus in the Attach to Data dialog with available tables and columns. If a Database Repository is found, information from your database will be merged with data from the repository file. If you deselect the **Get Tables and Columns from Database** checkbox Apama will no longer query your database for this information, but the Database Repository will still be used to populate drop down menus. Using a Database Repository to populate drop down menus makes it possible to specify which tables and columns from your database will be listed in the Attach to Data dialog and gives you the ability to build displays while databases are offline.

Apama includes a JDBC database driver for the following Apama-certified databases:

- DB2
- Microsoft SQL Server

- Oracle

These database drivers eliminate the need to install database-vendor-supplied drivers. The JDBC drivers can be used with any Apama component.

For more information on the supplied database drivers, see the documentation available in the following location:

```
apama_install_dir\doc\db_drivers\jdbc
```

## Direct JDBC connection

In order for Apama to communicate with your database using a straight JDBC connection, you must have a JDBC driver for your database.

Apama includes JDBC database drivers that eliminate the need to install database-vendor-supplied drivers. When you add a database to a dashboard you can specify the use of one of these Apama drivers. To add a database to a dashboard, see "SQL tab" on page 251, which provides information about the **Add Database** dialog. To use the Apama JDBC database driver for an added database, enter values for **JDBC Options** in the **Add Database** dialog. Also, be sure to add the `jar` file that contains the appropriate driver class to your **Dashboard Properties** (select **Properties** from the **Project** menu in Software AG Designer).

To use the Apama JDBC driver, specify the following according to the type of SQL database you want to add. In the URL, replace *HOSTNAME*, *PORT* and *DATABASENAME* or *DATABASESID* with the actual values for the particular database you want to connect to.

- **MSSQL** (`eysqlserver.jar` is in the *apama_install_dir*`\lib` folder)

  URL: `jdbc:sag:sqlserver://`*HOSTNAME*`::`*PORT*`;databaseName=`*DATABASENAME*

  Class name: `com.apama.jdbc.sqlserver.SQLServerDriver`

- **Oracle** (`eyoracle.jar` in the *apama_install_dir*`\lib` folder)

  URL: `jdbc:sag:oracle://`*HOSTNAME*`::`*PORT*`;SID=`*DATABASESID*

  Class name: `com.apama.jdbc.oracle.OracleDriver`

- **DB2** (`eydb2.jar` in the *apama_install_dir*`\lib` folder)

  URL: `jdbc:sag:db2://`*HOSTNAME*`::`*PORT*`;DatabaseName=`*DATABASENAME*

  Class name: `com.apama.jdbc.db2.DB2Driver`

JDBC drivers are available from most database vendors.

To make a non-Apama database driver available to Apama:

1. Locate the driver on your machine and add the `jar` that contains the driver class to your **Dashboard Properties** (select **Properties** from the **Project** menu in Software AG Designer).
2. Add the path to the JDBC driver `jar` file to the `APAMA_DASHBOARD_CLASSPATH` environment variable or to run the dashboard processes with `--dashboardExtraJars` option. This is required for the data server, display server or

dashboard builder to be able to find and load the JDBC driver class. You can add paths to multiple driver classes.

3. In the **Add Database** dialog, provide the database URL and the class name for your JDBC driver. The database URL typically contains the protocol and sub-protocol strings for your database as well as the path to the database and a list of properties. If you do not know the syntax for your database URL, consult the documentation for your JDBC driver.

## Setting SQL data source options

The Builder, Viewer, Data Server, and Display Server executables support the following command line option:

```
-q | --sql [retry:<ms> | fail:<n> | noinfo | nopererr | quote]
```

- `retry`: Specify the interval (in milliseconds) to retry connecting to a database after an attempt to connect fails. Default is `-1`, which disables this feature.
- `fail`: Specify the number of consecutive failed SQL queries after which to close this database connection and attempt to reconnect. Default is `-1`, which disables this feature.
- `noinfo`: Query database for available tables and columns in your database. If a Database Repository file is found, it is used to populate drop down menus in the Attach to SQL Data dialog.
- `nopererr`: SQL errors with the word **permission** in them will not be printed to the console. This is helpful if you have selected the **Use Client Credentials** option for a database. In this case, if your login does not allow access for some data in their display, you will not see any errors.
- `quote`: Encloses all table and column names specified in the Attach to SQL Data dialog in quotes when an SQL query is run. This is useful when attaching to databases that support quoted case-sensitive table and column names. *Note:* If a case-sensitive table or column name is used in the **Filter** field, or you are entering an advanced query in the **SQL Query** field, they must be entered in quotes, even if the `-sqlquote` option is specified.

# II Dashboard Property Reference for Graphs, Tables and Trends


- Introduction to Dashboard Properties ..... 261
- Graph Objects ..... 265
- Table Objects ..... 411
- Trend Objects ..... 453
- Drill-Down Specification ..... 535

Apama Dashboard Builder supports the design and deployment of dashboards that allow end users to visualize and interact with Apama scenarios and DataViews. Users of Dashboard Builder can incorporate a variety of visualization objects into their dashboards. This document provides reference information on those objects intended for the visualization of complex data: graphs, tables, and trend charts. Each visualization object is covered in a section that includes a complete listing of the object's properties.

# 10 Introduction to Dashboard Properties


- Objects for complex-data visualization ..... 262
- About the Object Properties window ..... 262
- Editing property values ..... 262
- Copying and pasting property values ..... 263

Users of Dashboard Builder can incorporate a variety of visualization objects into Apama dashboards, which allow end users to visualize and interact with Apama scenarios and DataViews. This document provides reference information on those objects intended for the visualization of complex data: graphs, tables, and trend charts. Each visualization object is covered in a section that includes a complete listing of the object's properties.

## Objects for complex-data visualization

This document covers the visualization objects contained in the following tabs of the Dashboard Builder **Object Palette**:

- **Graphs**: Bar graphs, heat maps, legends, pie graphs, radar graphs, and XY graphs. See "Graph Objects" on page 265.
- **Tables**: Standard tables and rotated tables. See "Table Objects" on page 411.
- **Trends**: Sparkline charts, stock charts, and trend graphs. See "Trend Objects" on page 453.

There is also an appendix on the Drill Down Properties dialog, which sets a drill-down-related property for all graph, table, and trend objects. See "Drill-Down Specification" on page 535.

## About the Object Properties window

To open the Object Properties window, select **Edit** | **Object Properties...** or click the Object Properties button on the toolbar. In the **Object Properties** window, you can view and edit the property values of an object selected in the Builder canvas area.

## Editing property values

Property names listed in the first column of the **Object Properties** panel cannot be changed. Property values, listed in the second column, can be set to static values or attached to dynamic data.

Blue text signifies that a property value is static and cannot be attached to a dynamic data source.

Green text signifies that a property value is currently attached to a dynamic data source and therefore it is no longer possible to edit this value directly in the **Object Properties** panel. See "Attaching Dashboards to Correlator Data" on page 55.

To remove a data attachment and restore the ability to edit property values directly in the **Object Properties** panel, right-click on the property name and select **Detach from Data** from the popup menu. An object property has been detached from the data source when the property name and value are no longer green.

## Copying and pasting property values

Copying and pasting makes it easy to transfer property values from one object to another.

There are two options for copying object properties:

- Copy all properties: To copy all object properties, both static properties and data attachments, select an object and click the copy button on the toolbar.
- Copy single property: To copy an individual property from the Object Properties window, right-click on the property name and select **Copy**. To copy a property from the Edit Function dialog, right-click in a text field and select **Copy**.

There are four options for pasting object properties:

- Paste data attachments: To paste only data attachments, select one or more objects and click on the paste data attachments button on the toolbar or use the keyboard shortcut **Ctrl+Shift+V**.

> **Note:** Only properties common to both objects are pasted onto the selected object or objects.

- Paste static properties: To paste only the static properties, select one or more objects and click on the paste static properties button on the toolbar. Note: Only properties common to both objects are pasted onto the selected object or objects.
- Paste all properties: To paste all properties, select one or more objects and click on the paste all properties button on the toolbar. This pastes all static properties as well as all data attachments. Note: Only properties common to both objects are pasted onto the selected object or objects.
- Paste single property: To paste an individual property into the **Object Properties** window, right-click on the property name and select **Paste**. The **Paste** option is enabled only if the copied attribute can be set on the selected property (data attachments, for example, cannot be pasted onto static properties). To paste a property in the **Edit Function** dialog, right-click in a text field and select **Paste**.

# 11 Graph Objects


- Bar graphs ..... 266
- Google map ..... 301
- Heat map ..... 309
- Legend ..... 329
- Pie graph ..... 338
- Radar graph ..... 353
- XY graph ..... 379

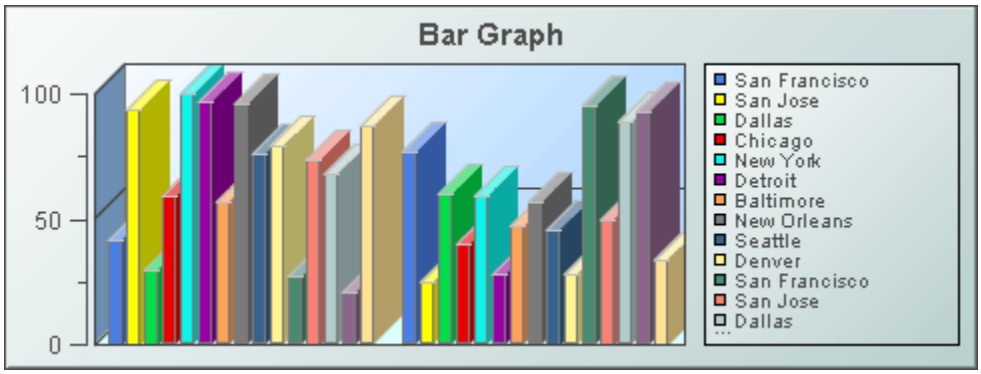
This chapter describes the visualization objects in the **Graphs** tab of the Dashboard Builder **Object Palette**.

## Bar graphs

Bar graphs visualize tabular data that has one or more numerical columns. Typically, the visualized data also has one non-numerical column, whose values are used as graph labels that uniquely identify each row.

A bar graph can visualize data in either of two ways:

- Row series visualization: One group of bars is shown for each numeric column in the data attachment. Within each group, there is a bar for each row in the data attachment.
- Column series visualization: one group of bars is shown for each row in your data attachment. Within each group, there is a bar for each numeric column in the data attachment.




Use the valueTable property to attach data to a bar graph. Use the rowSeriesFlag property to specify row series or column series visualization.

You can attach additional data to a bar graph by using the traceValueTable property. Data attached to this property is visualized with plotted points, or *trace markers*, rather than bars.

A bar graph can visualize trace data in either of two ways:

- Row series visualization: One group of trace markers is shown for each numeric column in the data attachment. Within each group, there is a marker for each row in the data attachment.
- Column series visualization: one group of trace markers is shown for each row in your data attachment. Within each group, there is a marker for each numeric column in the data attachment

The points within a group are connected to one another by a polyline, or *trace line*.

The current section covers the following kinds of bar graphs:

- **Bar graph**

- **3-D Stacked bar graph**
- **Grouped bar graph with traces**

These visualization objects all share the same properties. They differ from one another only with regard to their default values for these properties. When one of these objects is selected in the Builder canvas, the **Object Class Name** that appears at the top of the **Object Properties** pane is **obj_bargraph**.

The **Object Properties** panel organizes bar graph properties into the groups below.

## Bar graph: Alert group

Properties in this group allow you to specify changes in the appearance of bars, trace lines, and trace markers in response to changes in the status of plotted data elements. You can either specify threshold values (see valueHighAlarm, valueHighWarning, valueLowAlarm, and valueLowWarning) or attach a data table to traceValueAlarmStatusTable that indicates the status of each element of the table that is attached to traceValueTable.

### Alert group properties

This group includes the following properties:


- "traceValueAlarmStatusTable" on page 268
- "valueHighAlarm" on page 268
- "valueHighAlarmColor" on page 269
- "valueHighAlarmEnabledFlag" on page 269
- "valueHighAlarmLineVisFlag" on page 269
- "valueHighAlarmMarkColor" on page 269
- "valueHighAlarmMarkStyle" on page 269
- "valueHighWarning" on page 270
- "valueHighWarningColor" on page 270
- "valueHighWarningEnabledFlag" on page 270
- "valueHighWarningLineVisFlag" on page 270
- "valueHighWarningMarkColor" on page 270
- "valueHighWarningMarkStyle" on page 270
- "valueLowAlarm" on page 271
- "valueLowAlarmColor" on page 271
- "valueLowAlarmEnabledFlag" on page 271
- "valueLowAlarmLineVisFlag" on page 271

- "valueLowAlarmMarkColor" on page 271
- "valueLowAlarmMarkStyle" on page 272
- "valueLowWarning" on page 272
- "valueLowWarningColor" on page 272
- "valueLowWarningEnabledFlag" on page 272
- "valueLowWarningLineVisFlag" on page 272
- "valueLowWarningMarkColor" on page 273
- "valueLowWarningMarkStyle" on page 273


#### traceValueAlarmStatusTable

Attach an alarm table containing status indexes to this property in order to enable rule based alarm statuses for trace markers. The table attached to `traceValueAlarmStatusTable` must have the same number of rows and columns as traceValueTable. For each data element in traceValueTable, the status index at the corresponding position in `traceValueAlarmStatusTable` is used to set the alarm status of the marker that represents the data element.

Following are the valid indexes are:

- **0**: Use normal marker color and style. See "traceProperties" on page 297.
- **1**: Use low alarm marker color and style. See "valueLowAlarmMarkColor" on page 271 and "valueLowAlarmMarkStyle" on page 272.
- **2**: Use low warning marker color and style. See "valueLowWarningMarkColor" on page 273 and "valueLowWarningMarkStyle" on page 273.
- **3**: Use high warning marker color and style. See "valueHighWarningMarkColor" on page 270 and "valueHighWarningMarkStyle" on page 270.
- **4**: Use high alarm marker color and style. See "valueHighAlarmMarkColor" on page 269 and "valueHighAlarmMarkStyle" on page 269.
- **-1**: Determine marker color and style by comparing the value to the enabled alarm thresholds

If no data is attached to `traceValueAlarmStatusTable`, the alarm status for a trace marker is determined by comparing the marker's value to the enabled thresholds. See valueHighAlarm, valueHighWarning, valueLowAlarm, and valueLowWarning.

This property is in the **Alert** property group.

#### valueHighAlarm

Specifies the threshold value used by valueHighAlarmLineVisFlag, valueHighAlarmMarkColor, valueHighAlarmMarkStyle, and valueHighAlarmColor.

This property is in the **Alert** property group.

#### valueHighAlarmColor

When the value of a bar or trace segment is greater than or equal to valueHighAlarm, its color changes to the `valueHighAlarmColor`, provided valueHighAlarmEnabledFlag is selected.

This property is in the **Alert** property group.

#### valueHighAlarmEnabledFlag

Select to enable the high alarm threshold. See valueHighAlarm.

This property is in the **Alert** property group.

#### valueHighAlarmLineVisFlag

Select to display a dashed line at the high alarm threshold. The color of the line is set to valueHighAlarmMarkColor. This line is displayed only if valueHighAlarmEnabledFlag is selected.

This property is in the **Alert** property group.

#### valueHighAlarmMarkColor

When a trace marker's value is greater than or equal to valueHighAlarm, the marker changes to `valueHighAlarmMarkColor` and valueHighAlarmMarkStyle, provided valueHighAlarmEnabledFlag is selected and no data is attached to traceValueAlarmStatusTable.

If data is attached to traceValueAlarmStatusTable, a marker changes to `valueHighAlarmMarkColor` and valueHighAlarmMarkStyle when the marker's corresponding element in the attached alarm status table is 4.

If data is attached to traceValueAlarmStatusTable, and the marker's corresponding element in the attached alarm status table is -1, marker color and style behave as if no data were attached to traceValueAlarmStatusTable.

This property is in the **Alert** property group.

#### valueHighAlarmMarkStyle

When a trace marker's value is greater than or equal to valueHighAlarm, the marker changes to valueHighAlarmMarkColor and `valueHighAlarmMarkStyle`, provided valueHighAlarmEnabledFlag is selected and no data is attached to traceValueAlarmStatusTable.

If data is attached to traceValueAlarmStatusTable, a marker changes to valueHighAlarmMarkColor and `valueHighAlarmMarkStyle` when the marker's corresponding element in the attached alarm status table is 4.

If data is attached to traceValueAlarmStatusTable, and the marker's corresponding element in the attached alarm status table is -1, marker color and style behave as of no data were attached to traceValueAlarmStatusTable.

This property is in the **Alert** property group.

#### valueHighWarning

Specifies the threshold value used by valueHighWarningLineVisFlag, valueHighWarningMarkColor, valueHighWarningMarkStyle, and valueHighWarningColor.

This property is in the **Alert** property group.

#### valueHighWarningColor

When the value of a bar or trace segment is greater than or equal to valueHighWarning but less than valueHighAlarm, its color changes to `valueHighWarningColor`, provided valueHighWarningEnabledFlag is selected.

This property is in the **Alert** property group.

#### valueHighWarningEnabledFlag

Select to enable the high warning threshold. See valueHighWarning.

This property is in the **Alert** property group.

#### valueHighWarningLineVisFlag

Select to display a dashed line at the high warning threshold. The color of the line is set to valueHighWarningMarkColor. This line is displayed only if valueHighWarningEnabledFlag is selected.

This property is in the **Alert** property group.

#### valueHighWarningMarkColor

When a trace marker's value is greater than or equal to valueHighWarning but less than valueHighAlarm, the marker changes to `valueHighWarningMarkColor` and valueHighWarningMarkStyle, provided valueHighWarningEnabledFlag is selected and no data is attached to traceValueAlarmStatusTable.

If data is attached to traceValueAlarmStatusTable, a marker changes to `valueHighWarningMarkColor` and valueHighWarningMarkStyle when the marker's corresponding element in the attached alarm status table is 3.

If data is attached to traceValueAlarmStatusTable, and the marker's corresponding element in the attached alarm status table is -1, marker color and style behave as of no data were attached to traceValueAlarmStatusTable.

This property is in the **Alert** property group.

#### valueHighWarningMarkStyle

When a trace marker's value is greater than or equal to valueHighWarning but less than valueHighAlarm, the marker changes to valueHighWarningMarkColor and

`valueHighWarningMarkStyle`, provided valueHighWarningEnabledFlag is selected and no data is attached to traceValueAlarmStatusTable.

If data is attached to traceValueAlarmStatusTable, a marker changes to valueHighWarningMarkColor and `valueHighWarningMarkStyle` when the marker's corresponding element in the attached alarm status table is 3.

If data is attached to traceValueAlarmStatusTable, and the marker's corresponding element in the attached alarm status table is -1, marker color and style behave as of no data were attached to traceValueAlarmStatusTable.

This property is in the **Alert** property group.

#### valueLowAlarm

Specifies the threshold value used by valueLowAlarmLineVisFlag, valueLowAlarmMarkColor, valueLowAlarmMarkStyle, and valueLowAlarmColor.

This property is in the **Alert** property group.

#### valueLowAlarmColor

When the value of a bar or trace segment is less than or equal to valueLowAlarm, it's color changes to `valueLowAlarmColor`, provided valueLowAlarmEnabledFlag is selected.

This property is in the **Alert** property group.

#### valueLowAlarmEnabledFlag

Select to enable the low alarm threshold. See valueLowAlarm.

This property is in the **Alert** property group.

#### valueLowAlarmLineVisFlag

Select to display a dashed line at the low alarm threshold. The color of the line is set to valueLowAlarmMarkColor. This line is displayed only if valueLowAlarmEnabledFlag is selected.

This property is in the **Alert** property group.

#### valueLowAlarmMarkColor

When a trace marker's value is less than or equal to valueLowAlarm, the marker changes to `valueLowAlarmMarkColor` and valueLowAlarmMarkStyle, provided valueLowAlarmEnabledFlag is selected and no data is attached to traceValueAlarmStatusTable.

If data is attached to traceValueAlarmStatusTable, a marker changes to `valueLowAlarmMarkColor` and valueLowAlarmMarkStyle when the marker's corresponding element in the attached alarm status table is 1.

If data is attached to traceValueAlarmStatusTable, and the marker's corresponding element in the attached alarm status table is -1, marker color and style behave as of no data were attached to traceValueAlarmStatusTable.

This property is in the **Alert** property group.

#### valueLowAlarmMarkStyle

When a trace marker's value is less than or equal to valueLowAlarm, the marker changes to valueLowAlarmMarkColor and `valueLowAlarmMarkStyle`, provided valueLowAlarmEnabledFlag is selected and no data is attached to traceValueAlarmStatusTable.

If data is attached to traceValueAlarmStatusTable, a marker changes to valueLowAlarmMarkColor and `valueLowAlarmMarkStyle` when the marker's corresponding element in the attached alarm status table is 1.

If data is attached to traceValueAlarmStatusTable, and the marker's corresponding element in the attached alarm status table is -1, marker color and style behave as of no data were attached to traceValueAlarmStatusTable.

This property is in the **Alert** property group.

#### valueLowWarning

Specifies the threshold value used by valueLowWarningLineVisFlag, valueLowWarningMarkColor, valueLowWarningMarkStyle, and valueLowWarningColor.

This property is in the **Alert** property group.

#### valueLowWarningColor

When the value of a bar or trace segment is less than or equal to valueLowWarning but greater than valueLowAlarm, it changes to valueLowWarningColor, provided valueLowWarningEnabledFlag is selected.

This property is in the **Alert** property group.

#### valueLowWarningEnabledFlag

Select to enable the low warning threshold. See valueLowWarning.

This property is in the **Alert** property group.

#### valueLowWarningLineVisFlag

Select to display a dashed line at the low warning threshold. The color of the line is set to valueLowWarningMarkColor. This line is displayed only if valueLowWarningEnabledFlag is selected.

This property is in the **Alert** property group.

#### valueLowWarningMarkColor

When a trace marker's value is less than or equal to valueLowWarning but greater than valueLowAlarm, the marker changes to `valueLowWarningMarkColor` and valueLowWarningMarkStyle, provided valueLowWarningEnabledFlag is selected and no data is attached to traceValueAlarmStatusTable.

If data is attached to traceValueAlarmStatusTable, a marker changes to `valueLowWarningMarkColor` and valueLowWarningMarkStyle when the marker's corresponding element in the attached alarm status table is 2.

If data is attached to traceValueAlarmStatusTable, and the marker's corresponding element in the attached alarm status table is -1, marker color and style behave as of no data were attached to traceValueAlarmStatusTable.

This property is in the **Alert** property group.

#### valueLowWarningMarkStyle

When a trace marker's value is less than or equal to valueLowWarning but greater than valueLowAlarm, the marker changes to valueLowWarningMarkColor and `valueLowWarningMarkStyle`, provided valueLowWarningEnabledFlag is selected and no data is attached to traceValueAlarmStatusTable.

If data is attached to traceValueAlarmStatusTable, a marker changes to valueLowWarningMarkColor and `valueLowWarningMarkStyle` when the marker's corresponding element in the attached alarm status table is 2.

If data is attached to traceValueAlarmStatusTable, and the marker's corresponding element in the attached alarm status table is -1, marker color and style behave as of no data were attached to traceValueAlarmStatusTable.

This property is in the **Alert** property group.

## Bar graph: Background group

Properties in this group control the visibility and appearance of the portion of the graph that serves as the background of both the plot area and legend.

#### Background group properties

The group contains the following properties:

- "bgBorderColor" on page 274
- "bgBorderFlag" on page 274
- "bgColor" on page 274
- "bgEdgeWidth" on page 274
- "bgGradientColor2" on page 274
- "bgGradientMode" on page 274

- "bgRaisedFlag" on page 275
- "bgRoundness" on page 275
- "bgShadowFlag" on page 275
- "bgStyleFlag" on page 275
- "bgVisFlag" on page 275
- "borderPixels" on page 276


#### bgBorderColor

Sets the color of the border (see `bgBorderFlag`) of the background rectangle. Select the ... button and choose a color from the palette. Close the **Color Chooser** window when you are done.

#### bgBorderFlag

Select to display a border around the background rectangle.

This property is in the **Background** property group.

#### bgColor

Sets the background color. Select the ... button and choose a color from the palette. Close the **Color Chooser** window when you are done.

This property is in the **Background** property group.

#### bgEdgeWidth

Sets the width in pixels of the 3D edge on the background rectangle. This property is only used if `bgBorderFlag` is selected.

This property is in the **Background** property group.

#### bgGradientColor2

Sets the color for the second color in the gradient. The default is white. The `bgColor` property sets the first color in the gradient.

This property is in the **Background** property group.

#### bgGradientMode

Display a gradient in the background rectangle. Select from the following options:

- **None**: No gradient.
- **Diagonal Edge**: Gradient is drawn at a 45 degree angle from the top left to the bottom right corner of the object.
- **Diagonal Center**: Gradient is drawn at a 45 degree angle from the center to the top left and the bottom right corners of the object.

- **Horizontal Edge**: Gradient is drawn horizontally from the top to the bottom of the object.
- **Horizontal Center**: Gradient is drawn horizontally from the center to the top and bottom of the object.
- **Vertical Edge**: Gradient is drawn vertically from the left to the right of the object.
- **Vertical Center**: Gradient is drawn vertically from the center to the left and right of the object.

This property is in the **Background** property group.

#### bgRaisedFlag

Reverses the direction of the gradient, as well as that of the 3D edge if the `bgStyle` selected is 3D Rectangle.

This property is in the **Background** property group.

#### bgRoundness

Sets the arc length of the rounded corners. This property is only available if the `bgStyle` selected is **Round Rectangle**.

The value of `bgRoundness` cannot exceed half the value of the `objWidth` or the `objHeight`. If `bgRoundness` does exceed that value, half of `objWidth` or `objHeight` (whichever is smaller) will be used instead. For example if `objWidth` is `100` and `objHeight` is `50`, then the value of `bgRoundness` cannot exceed `25`. If it does, then half the value of `objHeight` (`25`) will be used instead. This property is in the **Background** property group.

#### bgShadowFlag

Select to display a drop shadow on the background rectangle.

This property is in the **Background** property group.

#### bgStyleFlag

Choose one of the following three options from the drop down menu:

- **Rectangle**: Select to display a background rectangle.
- **3D Rectangle**: Select to display a 3D edge on the background rectangle. If selected, use `bgEdgeWidth` to set the width of the 3D edge.
- **Round Rectangle**: Select to display a background rectangle with rounded edges. If selected, use `bgRoundness` to set the arc length of the rounded corners.

This property is in the **Background** property group.

#### bgVisFlag

Select to display the background rectangle.

This property is in the **Background** property group.

#### borderPixels

Sets the width in pixels of the border between the chart and the edge of the background rectangle.

This property is in the **Background** property group.

## Bar graph: Bar group

Properties in this group control the appearance of the graph's bars, including gradient style, color, and fill style. The group also includes properties that control the visibility and appearance of the text used to display bar values, including font, color, size, and position. You can also specify an image to be displayed within each bar.

#### Bar group properties

This group contains the following properties:


- "barGradientStyle" on page 276
- "barImage" on page 276
- "barProperties" on page 277
- "barValueTextColor" on page 278
- "barValueTextFont" on page 278
- "barValueTextHeight" on page 278
- "barValueTextPos" on page 278
- "barValueVisFlag" on page 279


#### barGradientStyle

Select one of the following in order to set the gradient style of the bars:

- **None**: Default setting.
- **Shaded**: Display bars with a flat gradient
- **Rounded**: Display bars with a rounded gradient

This property is in the **Bar** property group.

#### barImage

Specifies an image (`.gif`, `.jpg`, or `.png` file) to display in each bar. Select the name of the image file from the drop down menu, or enter the pathname of the file. The drop down menu contains the names of image files located in the current directory (typically, the `dashboards` directory of your project directory, under your Apama installation's work directory), as well as image files located in the first level of subdirectories. If you

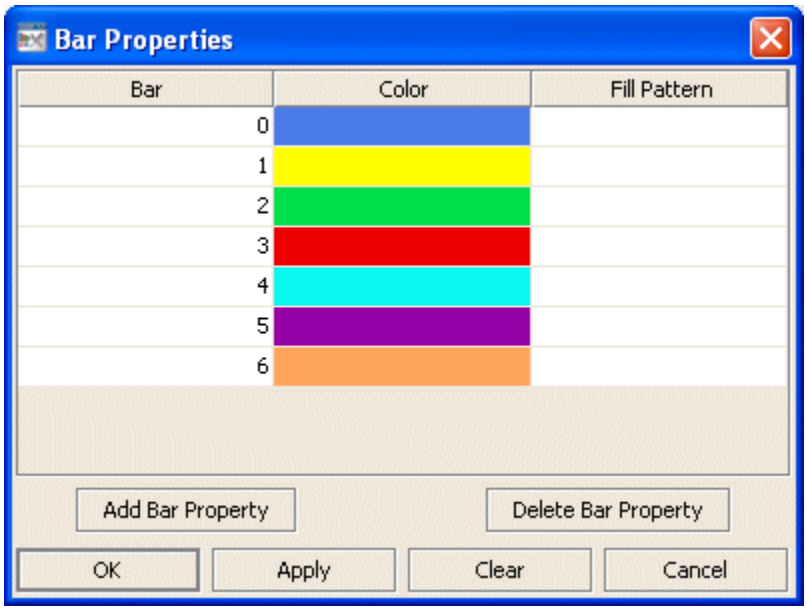
enter a pathname, use an absolute pathname or a pathname that is relative to the current directory.

**Note:** If necessary, the image will be stretched to fit the bar size.

This property is in the **Bar** property group.

#### barProperties

Specifies the color and fill pattern for each bar in the graph. In the **Object Properties** window, double-click on `barProperties` in the **Property Name** field to bring up the **Bar Properties** dialog. In the **Bar Properties** dialog you can assign attributes to each bar in a bar graph.




The dialog contains three columns of fields:

- **Bar**: There is one entry for each bar that is currently displaying data in the bar graph. The Color and Fill Pattern columns list the current settings for each bar.
- **Color**: Select the ellipsis button in the Color column and choose a color from the palette to set the color of the bar. Close the Color Chooser window.
- **Fill Pattern**: Select the ellipsis button in the Fill Pattern column and choose a pattern from the palette to set the fill pattern of the bar. Close the Fill Pattern window.

**Note:** The fill patterns in your bar graph are ignored unless the barGradientStyle property is set to **None**.

The dialog contains the following buttons:

- **Add Bar Property**: Click to add a Bar Property entry. This does not add a bar to your graph, it adds a bar entry so that you can set properties for bars that will display data that is not yet available. This is useful if the data attachment is not available

when you setup your bar graph, or if the number of rows or columns returned by your data attachment varies.

- **Delete Bar Property**: Removes the last bar property entry from the **Bar Properties** dialog.
- **OK**: Applies values and closes the dialog.
- **Apply**: Applies values without closing the dialog.
- **Reset**: Resets all fields to last values applied.
- **Clear**: Clears all fields. Detaches object from data source (once Apply or OK is selected).
- **Cancel**: Closes the dialog with last values applied.

This property is in the **Bar** property group.

#### barValueTextColor

Sets the color of the text used to display bar values. This property is visible in the Builder's **Object Properties** pane only if barValueVisFlag is selected.

To set the color, select the ... button and choose a color from the palette. Close the **Color Chooser** window when you are done.

This property is in the **Bar** property group.

#### barValueTextFont

Sets the font of the text used to display bar values. This property is visible in the Builder's **Object Properties** pane only if barValueVisFlag is selected. To set the font, select an item from the drop down list.

This property is in the **Bar** property group.

#### barValueTextHeight

Sets the point size of the text used to display bar values. This property is visible in the Builder's **Object Properties** pane only if barValueVisFlag is selected. To set the point size, enter a number in the text field.

This property is in the **Bar** property group.

#### barValueTextPos

Sets the position, relative to the bar, of the text used to display bar values. This property is visible in the Builder's **Object Properties** pane only if barValueVisFlag is selected. To set the text position, choose an item from the drop down list. By default, digits after a decimal point are not displayed on the labels.

This property is in the **Bar** property group.

#### barValueVisFlag

Select to display a value for each bar. Selecting this property causes the following properties to appear in the **Object Properties** panel:

- barValueTextColor
- barValueTextFont
- barValueTextHeight
- barValueTextPos

This property is in the **Bar** property group.

## Bar graph: Column group

This group contains one property, columnsToHide, which controls which data-attachment columns are used for plotted data or labels.

#### columnsToHide

Specifies columns from the data attachment to exclude from being used for plotted data or labels. Data from the labelColumnName column will be used for labels even if that column name is also specified in the `columnsToHide` property. Columns specified in the `columnsToHide` property can still be used in the `drillDownColumnSubs` property.

This property is in the **Column** property group.

## Bar graph: Data group

Properties in this group control what data appears in the graph, and whether the data appears in column series or row series form.

#### Data group properties

The group contains the following properties:


- "rowSeriesFlag" on page 280
- "traceValueDivisor" on page 280
- "traceValueTable" on page 280
- "traceYAxisValueMax" on page 281
- "traceYAxisValueMin" on page 281
- "valueDivisor" on page 281
- "valueTable" on page 281
- "yValueMax" on page 282
- "yValueMin" on page 282

#### rowSeriesFlag

This property controls how row and column data populate the graph:

- If the rowSeriesFlag checkbox is selected, one group of bars is shown for each numeric column in your data attachment. Within each group, there is a bar for each row in the data attachment.

If xAxisFlag is enabled, each group is labeled with the name of the corresponding numeric column.

By default, each bar within a group has a different color. If rowLabelVisFlag is selected, the legend indicates the mapping between each bar's color and the label-column value of the bar's corresponding row (see labelColumnName). If both rowLabelVisFlag and rowNameVisFlag are deselected, the legend indicates the mapping between each bar's color and an integer identifier for the bar's corresponding row.

- If the rowSeriesFlag checkbox is not selected, one group of bars is shown for each row in your data attachment. Within each group, there is a bar for each numeric column in the data attachment.

If both rowLabelVisFlag and xAxisFlag are enabled, each group is labeled with the label-column value for the group's corresponding row (see labelColumnName). If rowLabelVisFlag is disabled and xAxisFlag is enabled, each group is labeled with an integer identifier for the group's corresponding row.

By default, each bar within a group has a different color. The legend indicates the mapping between each bar's color and the name of the bar's corresponding numeric column.

This property is in the **Data** property group.

#### traceValueDivisor

Divides trace values by the number entered.

The default is 1.

This property is in the **Data** property group.

#### traceValueTable

Attach your data to the `traceValueTable` property to add one or more traces to your bar graph. Right-click on the property name in the Object Properties panel, and select a menu item under **Attach to Data**. The attached data table should have one or more numerical columns. Typically, the data attachment has one non-numerical column, whose values uniquely identify each row (that is, no two rows of the table have the same value for the non-numerical column).

The property rowSeriesFlag controls how row and column data populate the graph:

- If the rowSeriesFlag checkbox is selected, one trace line is shown for each row your data attachment. Within each trace line, there is a mark for each numeric column in the data attachment. The height of a given mark in a given trace line is proportional

to the value of the mark's corresponding numerical column for the trace line's corresponding row.

By default, each trace line has a different color. If rowLabelVisFlag is selected, the legend indicates the mapping between each line's color and the trace-label-column value of the bar's corresponding row (see traceLabelColumnName).

- If the rowSeriesFlag checkbox is not selected, one trace line is shown for each numeric column in your data attachment. Within each trace line, there is a mark for each row of the data attachment. The height of a given mark in a given trace line is proportional to the value of the trace line's corresponding numerical column for the mark's corresponding row.

By default, each trace line has a different color. The legend indicates the mapping between each trace line's color and the name of the trace line's corresponding numeric column.

This property is in the **Data** property group.

#### traceYAxisValueMax

When traceYAxisFlag is selected, the traceYAxisValueMin and `traceYAxisValueMax` properties are used to control the range of the trace y-axis if yAxisAutoScaleMode is set to **Off** or **On-include Min/Max**.

This property is in the **Data** property group.

#### traceYAxisValueMin

When traceYAxisFlag is selected, the `traceYAxisValueMin` and traceYAxisValueMax properties are used to control the range of the trace y-axis if yAxisAutoScaleMode is set to **Off** or **On-include Min/Max**.

This property is in the **Data** property group.

#### valueDivisor

Divides bar and y-axis values by the number entered.

The default is 1.

This property is in the **Data** property group.

#### valueTable

Attach your data to the `valueTable` property. Right-click on the property name in the Object Properties panel, and select a menu item under **Attach to Data**. The attached data table should have one or more numerical columns. Typically, the data attachment has one non-numerical column, whose values uniquely identify each row (that is, no two rows of the table have the same value for the non-numerical column).

The property rowSeriesFlag controls how row and column data populate the graph:

- If the rowSeriesFlag checkbox is selected, one group of bars is shown for each numeric column in your data attachment. Within each group, there is a bar for

each row in the data attachment. The height of a given bar in a given group is proportional to the value of the group's corresponding numerical column for the bar's corresponding row.

If xAxisFlag is enabled, each group is labeled with the name of the corresponding numeric column.

By default, each bar within a group has a different color. If rowLabelVisFlag is selected, the legend indicates the mapping between each bar's color and the label-column value of the bar's corresponding row (see labelColumnName). If both rowLabelVisFlag and rowNameVisFlag are deselected, the legend indicates the mapping between each bar's color and an integer identifier for the bar's corresponding row.

- If the rowSeriesFlag checkbox is not selected, one group of bars is shown for each row in your data attachment. Within each group, there is a bar for each numeric column in the data attachment. The height of a given bar in a given group is proportional to the value of the bar's corresponding numerical column for the group's corresponding row.

If both rowLabelVisFlag and xAxisFlag are enabled, each group is labeled with the label-column value for the group's corresponding row (see labelColumnName). If rowLabelVisFlag is disabled and xAxisFlag is enabled, each group is labeled with an integer identifier for the group's corresponding row.

By default, each bar within a group has a different color. The legend indicates the mapping between each bar's color and the name of the bar's corresponding numeric column.

This property is in the **Data** property group.

#### yValueMax

The yValueMin and `yValueMax` properties control the range of the y-axis if the yAxisAutoScaleMode is set to Off. In addition, if yAxisAutoScaleMode is set to **On - Include Min/Max**, the dashboard calculates the smallest y-axis range that includes both yValueMin and `yValueMax` as well as all plotted points.

This property is in the **Data** property group.

#### yValueMin

The `yValueMin` and yValueMax properties control the range of the y-axis if the yAxisAutoScaleMode is set to Off. In addition, if yAxisAutoScaleMode is set to **On - Include Min/Max**, the dashboard calculates the smallest y-axis range that includes both `yValueMin` and yValueMax as well as all plotted points.

This property is in the **Data** property group.

## Bar graph: Data Format group

Properties on this group control the format of displayed bar values as well as numerical and date labels.

#### Data Format group properties

The group includes the following properties:

- "labelColumnFormat" on page 283
- "traceYValueFormat" on page 283
- "yValueFormat" on page 283

#### labelColumnFormat

Sets the format of numeric or date labels displayed on the x-axis, in the legend, and in tooltips.

Select or enter the format specification. Use syntax from the Java `DecimalFormat` class for numeric labels, and syntax from the Java `SimpleDateFormat` class for date labels.

To enable tooltips, select the mouseOverFlag.

This property is in the **Data Format** property group.

#### traceYValueFormat

Sets the numeric format of bar values displayed in the legend and in tooltips.

Select or enter a format. Use syntax from the Java `DecimalFormat` class. To enable tooltips, select the mouseOverFlag.

This property is in the **Data Format** property group.

#### yValueFormat

Sets the numeric format of bar values displayed on bars, in the legend and in tooltips

Select or enter a format. Use syntax from the Java `DecimalFormat` class. To enable tooltips, select the mouseOverFlag.

This property is in the **Data Format** property group.

## Bar graph: Data Label group

Properties in this group control the labels that are used along the x-axis or in the legend.

#### Data Label group properties

The group contains the following properties:

- "columnDisplayNames" on page 284
- "labelColumnName" on page 284
- "rowLabelVisFlag" on page 284
- "rowNameVisFlag" on page 284

- "traceLabelColumnName" on page 285

#### columnDisplayNames

Sets alternate display names for the columns of the data attached to valueTable. Column names are displayed either along the x-axis or in the legend, depending on whether or not the rowSeriesFlag is selected.

This property is in the **Data Label** property group.

#### labelColumnName

Sets the label column. By default, the label column is the first non-numeric text column in your data attachment, if there is one. Data from the label column is used to label either the x-axis or the legend, depending on whether rowSeriesFlag is enabled.

If both rowSeriesFlag and rowLabelVisFlag are enabled, data from the label column will be used in the legend.

If rowSeriesFlag is not enabled and both rowLabelVisFlag and xAxisFlag are enabled, data from the label column will appear on the x-axis.

This property is in the **Data Label** property group.

#### rowLabelVisFlag

Determines whether or not data from the label column is used in chart labels. (By default, the label column is the first non-numeric column in your data attachment. You can override this default with labelColumnName.)

If both rowSeriesFlag and rowLabelVisFlag are enabled, data from the label column is used in the legend. If rowSeriesFlag is enabled and both rowLabelVisFlag and rowNameVisFlag are disabled, integer row identifiers are used in the legend.

If rowSeriesFlag is not enabled and both rowLabelVisFlag and xAxisFlag are enabled, data from the label column will appear on the x-axis. If rowSeriesFlag, rowLabelVisFlag, and rowNameVisFlag are disabled, and xAxisFlag is enabled, integer row identifiers are used on the x-axis.

This property is in the **Data Label** property group.

#### rowNameVisFlag

Determines whether generated row names are used in chart labels. Enable this property if your data attachment has no label column (see labelColumnName). Note that if both rowNameVisFlag and rowLabelVisFlag are enabled, row names and label-column values can appear side-by-side in chart labels.

This property is in the **Data Label** property group.

#### traceLabelColumnName

Sets the trace label column. By default, the trace label column is the first non-numeric text column in your data attachment. Data from the label column is used in the legend, if rowSeriesFlag is enabled.

If both rowSeriesFlag and rowLabelVisFlag are enabled, data from the label column will be used in the legend.

This property is in the **Data Label** property group.

## Bar graph: Historian group

Do not use the properties in this group.

#### historyTableName

Do not use this property.

This property is in the **Historian** property group.

#### historyTableRowNameFlag

Do not use this property.

This property is in the **Historian** property group.

## Bar graph: Interaction group

Properties in this group control various forms of interaction between the end user and the graph, including scrolling, highlighting, and activating commands, drill downs, and tooltips.

#### Interaction group properties

The group includes the following properties:


- "command" on page 286
- "commandCloseWindowOnSuccess" on page 286
- "commandConfirm" on page 286
- "commandConfirmText" on page 287
- "drillDownColumnSubs" on page 287
- "drillDownSelectMode" on page 287
- "drillDownTarget" on page 288
- "mouseOverFlag" on page 288
- "mouseOverHighlightFlag" on page 288
- "scrollbarMode" on page 288

- "scrollbarSize" on page 288

#### command

Assign a command or group of commands to this stock chart by right-clicking on the `command` property name in the **Object Properties** window. Select **Define Command** and choose **SYSTEM**, **APAMA**, or **MULTIPLE**. See "Using the Define Apama Command dialog" on page 226.

Once a command or command group has been assigned to this object, you can activate it from a deployed dashboard or from the Dashboard Builder:

- Dashboard Builder: Double click on the object.
- Web-based deployment: Single click on the object or else right click on it and select **Execute Command** from the popup menu.
- Local deployment: By default, single-click on the object or else right-click on it and select **Execute Command** from the popup menu. To override the default, select **Tools > Options** in the Builder (do this before you generate the deployment package), and uncheck **Single-Click for Drill Down and Commands** in the **General** tab. This allows the end user to use either a double click or a right click.

When you activate a command, any defined drill down substitutions are performed, and then the command is executed.

If you assign multiple commands, the commands are launched in an arbitrary order, and are executed asynchronously; there is no guarantee that one command will finish before the next one in the sequence starts.

This property is in the **Interaction** property group.

#### commandCloseWindowOnSuccess

Select this property to automatically close the window that initiates a **SYSTEM** command when the command is executed successfully. This applies to **SYSTEM** commands only, and is not supported at all for thin-client, Web-page deployments.

With **APAMA** commands, the window is closed whether or not the command is executed successfully. For **MULTIPLE** commands, the window closes when the first command in the command group succeeds.

This property is in the **Interaction** property group.

#### commandConfirm

By default, when the end user executes a command (see the `command` property), the command confirmation dialog is disabled. To control this option for each individual object, use the `commandConfirm` check box. If confirmation is required for a **MULTIPLE** command group, a single confirmation dialog is presented; if you confirm the execution, all individual commands in the group are executed with no further confirmation. If the you cancel the execution, none of the commands in the group is executed.

You can also override the confirmation status of individual objects with an application-wide policy. Select **Tools | Options** and choose from three confirmation values:

- **Do not confirm**: Indicates that no commands require confirmation (regardless of each object's confirmation status).
- **Confirm all**: Indicates that all commands require confirmation (regardless of each object's confirmation status).
- **Use object confirm flag** (default): Indicates that the confirmation status of each object will determine whether confirmation is required.

This property is in the **Interaction** property group.

#### commandConfirmText

Use this property to write your own text for the confirmation dialog. Otherwise, default text is used. See `commandConfirm`.

This property is in the **Interaction** property group.

#### drillDownColumnSubs

Use this property to direct a dashboard to assign data-table column values to specified dashboard variables when the end user activates a drilldown on this object. In the Object Properties window, double-click on `drillDownColumnSubs` in the **Property Name** field to bring up the Drill Down Column Substitutions dialog.

The dialog has the following fields and buttons:

- **Substitution String**: Enter the dashboard variable next to the name of the data table column whose value you want assigned to the variable. Press Enter.
- **Add Column**: Enter the name of a column and click the **Add Column** button to insert a column into the table.
- **Clear**: Click the **Clear** button to remove all variables listed.

The **Column Name** list is populated based on the table's data attachment. If you have not yet attached the table to data, this list is empty.

Once you have selected which column values to pass in as substitutions, double-click on any element in your object to open a drill down window that displays corresponding values.

This property is in the **Interaction** property group.

#### drillDownSelectMode

Use this property to control how a drill down display is activated. Select one of the following:

- **Anywhere** to activate a drill down display by double-clicking anywhere on the chart.

- **Element Only** to enable a drill down display only when you double-click on an element of the chart, such as a bar or candlestick.

This property is in the **Interaction** property group.

#### drillDownTarget

To specify a drill down display, double click on `drillDownTarget` in the **Property Name** field to bring up the Drill Down Properties dialog. See "Drill-Down Specification" on page 535.

This property is in the **Interaction** property group.

#### mouseOverFlag

Select this property to enable tooltips for your bar graph. To display a tooltip, point to a bar or trace marker with your mouse. The tooltip will contain information from your data attachment about that bar or marker.

This property is in the **Interaction** property group.

#### mouseOverHighlightFlag

Select this property to enable bar highlighting. To highlight a bar in red, point to the bar.

This property is in the **Interaction** property group.

#### scrollbarMode

Select one of the following to set the behavior of the x-axis scroll bar in the graph:

- **Never**: Default setting. Some bars may get clipped.
- **As Needed**: Display the scroll bar when there is not enough space to display all of the bars in the plot area. Each bar uses at least `minSpacePerBar` pixels along the x-axis.
- **Always**: Display a scroll bar at all times.

**Note:** If drawHorizontalFlag is selected, the x-axis is vertical.

This property is in the **Interaction** property group.

#### scrollbarSize

Specify the height of the horizontal scroll bar and the width of the vertical scroll bar, in pixels.

The default value is -1, which sets the size to the system default.

This property is in the **Interaction** property group.

## Bar graph: Label group

Properties in this group control the graph's main label (which defaults to **Bar Graph**), including text, alignment, color, font, and size.

### Label group properties

The group includes the following properties:


- "label" on page 289
- "labelTextAlignX" on page 289
- "labelTextColor" on page 289
- "labelTextFont" on page 289
- "labelTextHeight" on page 289


#### label

Specifies the text for the chart label. Click the ellipsis for multi-line text.

The default is **Bar Graph**.

This property is in the **Label** property group.

#### labelTextAlignX

Sets the alignment of the chart label (see the label property). Select **Left**, **Center**, or **Right** from the drop down list.

This property is in the **Label** property group.

#### labelTextColor

Specifies the color of the chart label text (see the label property). Select the **...** button and choose a color from the palette. Close the Color Chooser window when you are done.

This property is in the **Label** property group.

#### labelTextFont

Specifies the font of the chart label text (see the label property). Select an item from drop down list.

This property is in the **Label** property group.

#### labelTextHeight

Specifies the point size of the chart label text (see the label property).

This property is in the **Label** property group.

## Bar graph: Layout group

Properties in this group control the layout of bars and axis labels, including alignment and spacing of bars, as well as spacing and rotation of axis labels. You can also specify 3D bars, as well as a horizontal, stacked, or waterfall arrangement for the bars.

### Layout group properties

This group contains the following properties:


- "barCenterFlag" on page 290
- "barFitFlag" on page 290
- "draw3dDepth" on page 290
- "draw3dFlag" on page 291
- "drawHorizontalFlag" on page 291
- "drawStackedFlag" on page 291
- "drawWaterfallFlag" on page 291
- "horizAxisLabelRotationAngle" on page 291
- "horizAxisMinLabelHeight" on page 291
- "minSpaceBetweenBars" on page 291
- "minSpaceBetweenGroups" on page 291
- "minSpacePerBar" on page 292


#### barCenterFlag

Select to center the bars in the plot area. If not selected, the bars will be left or top aligned, depending on drawHorizontalFlag. This property is only used if the barFitFlag is not selected.

This property is in the **Layout** property group.

#### barFitFlag

Select to stretch the bars to fit the available space in the plot area. If deselected, the minSpacePerBar property is used to determine the bar width.

This property is in the **Layout** property group.

#### draw3dDepth

Sets the depth in pixels of the bars, provided draw3dFlag is enabled.

This property is in the **Layout** property group.

#### draw3dFlag

Select to change the display of the bars from 2D to 3D.

This property is in the **Layout** property group.

#### drawHorizontalFlag

Select to have the bars in your graph displayed horizontally.

This property is in the **Layout** property group.

#### drawStackedFlag

Select to stack each bar group in your graph.

This property is in the **Layout** property group.

#### drawWaterfallFlag

Select to stack each bar group in your graph with an offset between bar sections.

This property is in the **Layout** property group.

#### horizAxisLabelRotationAngle

Sets the amount of rotation of labels on the horizontal axis. Values range from 0 to 90 degrees. A value of 0 causes the bar graph to automatically pick the optimum angle of rotation (this is the default).

This property is in the **Layout** property group.

#### horizAxisMinLabelHeight

Sets the minimum amount of space to reserve for labels on the horizontal axis. If axis labels vary over time, this property can be used to reserve a consistent amount of space to prevent overlapping.

This property is in the **Layout** property group.

#### minSpaceBetweenBars

Set the minimum space between bars, in pixels.

This property is in the **Layout** property group.

#### minSpaceBetweenGroups

Set the minimum space between bar groups, in pixels.

This property is in the **Layout** property group.

#### minSpacePerBar

Sets the minimum width for each bar, in pixels, provided drawHorizontalFlag is disabled.

The default value is 1.

This property is in the **Layout** property group.

#### vertAxisMinLabelWidth

Specifies the minimum width in pixels for the vertical axis labels.

This property is in the **Layout** property group.

#### waterfallBarConnectFlag

If drawWaterfallFlag is checked, select to connect the bar sections in each bar group.

This property is in the **Layout** property group.

#### waterfallTotalBarColor

Specifies the color for the bar that shows the sum of the bar sections in each bar group. See waterfallTotalFlag.

This property is in the **Layout** property group.

#### waterfallTotalBarFStyle

Do not use this property.

This property is in the **Layout** property group.

#### waterfallTotalBarLabel

Specifies the label for the bar that shows the sum of the bar sections in each bar group. See waterfallTotalFlag.

This property is in the **Layout** property group.

#### waterfallTotalFlag

If drawWaterfallFlag is checked, select to display a bar that shows the sum of the bar sections in each bar group.

This property is in the **Layout** property group.

## Bar graph: Legend group

Properties in this group control the visibility, appearance, and content of the graph legend.

#### Legend group properties

The group contains the following properties:

- "legendBgColor" on page 293
- "legendBgGradientFlag" on page 293
- "legendValueVisFlag" on page 293
- "legendVisFlag" on page 293
- "legendWidthPercent" on page 293

#### legendBgColor

Specifies the background color of the legend. Select the **...** button and choose a color from the palette. Close the Color Chooser window when you are done.

This property is in the **Legend** property group.

#### legendBgGradientFlag

Select to display a gradient in the legend background.

This property is in the **Legend** property group.

#### legendValueVisFlag

Select to display the numerical values of your data in the legend.

This property is in the **Legend** property group.

#### legendVisFlag

Select to display the legend.

This property is in the **Legend** property group.

#### legendWidthPercent

Set the percent of the total width of the object used for the legend.

This property is in the **Legend** property group.

### Bar graph: Marker group

Properties in this group control the appearance of trace markers (but see also "Bar graph: Trace group" on page 297).

#### Marker group properties

The group contains the following properties:

- "markDefaultSize" on page 294

- "markScaleMode" on page 294

#### markDefaultSize

Sets the size of the markers (see "traceProperties" on page 297) in pixels. Supply an integer value that is between 1 and 18, inclusive.

This property is in the **Marker** property group.

#### markScaleMode

Sets the scale mode for trace marks. Select one of the following from the drop down menu:

- **No Scale**: All marks, across and within traces, are the same size.
- **Scale by Trace**: Scale marks according to the trace in which they reside, that is, marks in the first trace are the largest, across all traces, and the marks in the last trace are the smallest.
- **Scale Within Trace**: Scale marks according to the relative order of the data within each trace.

This property is in the **Marker** property group.

## Bar graph: Object group

Properties in this group control the visibility and transparency of the graph as a whole. They also control (or reflect) the overall position and dimensions of the graph. In addition, a property in this group reflects the generated name of this individual graph.

#### Object group properties

This group contains the following properties:

- "anchor" on page 295
- "dock" on page 295
- "objHeight" on page 295
- "objName" on page 295
- "objWidth" on page 295
- "objX" on page 295
- "objY" on page 295
- "transparencyPercent" on page 296
- "visFlag" on page 296

#### anchor

Select zero or more of **Top**, **Left**, **Bottom**, and **Right** to control the object's placement. The `anchor` property is only applied when the display is resized either by changing the **Background Properties** on the display or by resizing the window in **Layout** mode. If an object has the `dock` property set, the `anchor` property is ignored. See "About resize modes" on page 39.

#### dock

Select **None** (default), **Top**, **Left**, **Bottom**, **Right**, or **Fill** in order to control the object's placement in **Layout** resize mode. See "About resize modes" on page 39.

#### objHeight

Set the height of a chart by entering a value for this property or by dragging a handle of the bounding box that appears when the chart is selected. When you drag a handle of the bounding box, the displayed value for this property changes to reflect the real-time height of the chart.

This property is in the **Object** property group.

#### objName

An identifier that is generated by the Dashboard Builder. This name can be used by other objects' properties in order to refer to the named chart, as with, for example, the graphName property of the Legend visualization object.

This property is in the **Object** property group.

#### objWidth

Set the width of a chart by entering a value for this property or by dragging a handle of the bounding box that appears when the chart is selected. When you drag a handle of the bounding box, the displayed value for this property changes to reflect the real-time width of the chart.

This property is in the **Object** property group.

#### objX

Sets the X coordinate of the center of this visualization object, relative to the lower left corner of the current dashboard. This value is set automatically when you position the object with the mouse.

This property is in the **Object** property group.

#### objY

Sets the Y coordinate of the center of this visualization object, relative to the lower left corner of the current dashboard. This value is set automatically when you position the object with the mouse.

This property is in the **Object** property group.

#### transparencyPercent

Sets the transparency of this chart.

This property is in the **Object** property group.

#### visFlag

Deselect to make this visualization object invisible in the current dashboard.

This property is in the **Object** property group.

## Bar graph: Plot Area group

Properties in this group control the appearance of the plot area, the rectangular area that serves as background for the bars (but not for the legend or axis labels; see "Bar graph: Background group" on page 273).

#### Plot Area group properties

The group includes the following properties:

- "gridBgColor" on page 296
- "gridBgGradientFlag" on page 296
- "gridBgImage" on page 296
- "gridColor" on page 297
- "traceFillStyle" on page 297

#### gridBgColor

Sets the color of the plot area. Select the **...** button and choose a color from the palette. Close the Color Chooser window when you are done.

This property is in the **Plot Area** property group.

#### gridBgGradientFlag

Select to display a gradient in the grid background. Set the color of the grid background with the labelTextAlignX property.

This property is in the **Plot Area** property group.

#### gridBgImage

Specify an image (`.gif`, `.jpg`, or `.png` file) to display in the plot area. Select the name of the image file from the drop down menu, or enter the pathname of the file. The drop down menu contains the names of image files located in the current directory (typically, the `dashboards` directory of your project directory, under your Apama installation's work directory), as well as image files located in the first level of subdirectories. If you

enter a pathname, use an absolute pathname or a pathname that is relative to the current directory.

This property is in the **Plot Area** property group.

#### gridColor

Sets the color of the horizontal line or lines in the plot area that mark y-axis major divisions. Select the **...** button and choose a color from the palette. Close the Color Chooser window when you are done.

This property is in the **Plot Area** property group.

## Bar graph: Trace group

Properties in this group control the appearance of trace lines and trace markers (but see also "Bar graph: Marker group" on page 293), including color, style, and line width.

#### Trace group properties

This group includes the following properties:

- "traceFillStyle" on page 297
- "traceProperties" on page 297
- "traceShadowFlag" on page 299

#### traceFillStyle

Set `traceFillStyle` to one of the following fill styles for the area under the trace:

- **Solid**
- **Transparent**
- **Gradient**
- **Transparent Gradient**
- **None**

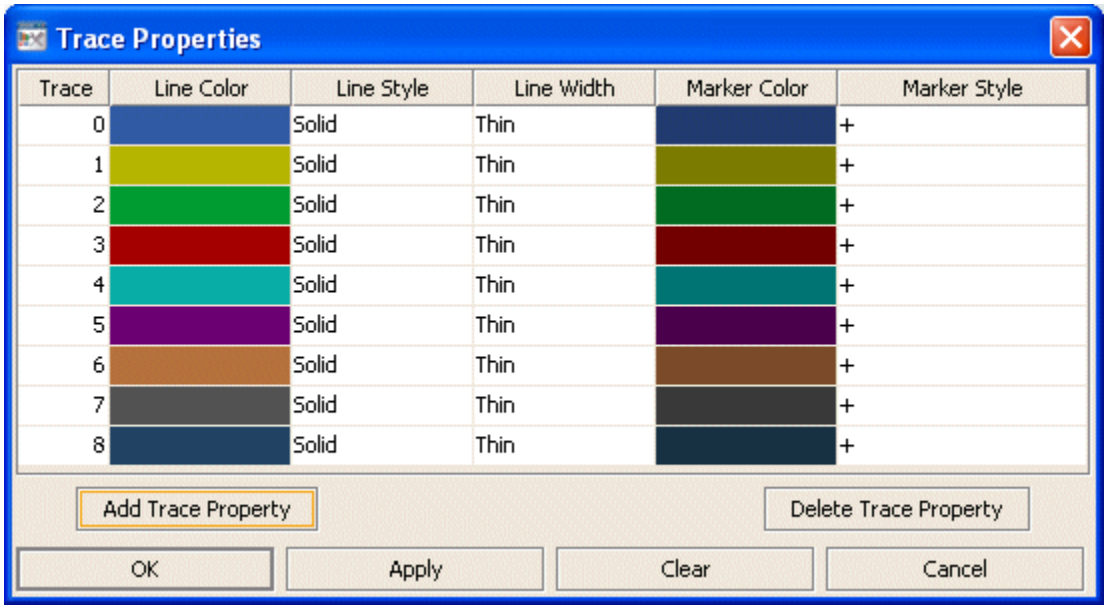
None is the default.

This property is in the **Trace** property group.

#### traceProperties

Specify the line color, line style, line width, marker color and marker style of all traces.

In the Object Properties window, double-click on `traceProperties` in the **Property Name** field to bring up the Trace Properties dialog. In the Trace Properties dialog you can assign attributes to each plotting trace in your graph.




The dialog has six columns of fields:

- **Trace**: One field for each trace that is currently in the graph. Current settings for each trace are shown.
- **Line Color**: Select the ellipsis button in the **Color** column and choose a color from the palette. Close the Color Chooser window.
- **Line Style**: Select the ellipsis button in the **Line Style** column and choose a style from the drop down menu. Choose either **No Line**, **Solid**, **Dotted**, **Dashed**, or **Dot Dashed**.
- **Line Width**: Select the ellipsis button in the **Line Width** column and choose a size from the drop down menu. Choose either **Thin**, **Medium** or **Thick**.
- **Marker Color**: Select the ellipsis button in the **Marker Color** column and choose a color from the palette. Close the **Color Chooser** window.
- **Marker Style**: Select the ellipsis button in the **Marker Style** column and choose a style from the drop down menu. Choose either **No Marker**, **Dot**, **+**, *****, **o**, **x**, **Filled Circle**, **Filled Diamond**, **Filled Triangle**, **Filled Square**, or **Filled Star**.

The dialog contains the following buttons:

- **Add Trace Property**: Click to add a trace property field. The data for the trace does not have to be available yet. You may consider adding and assigning attributes to more traces than your data currently needs for when you have more data to show. It is not necessary to set properties for each trace you currently or subsequently have. This is optional and can be done after additional data is displayed in a subsequent new trace.
- **Delete Trace Property**: Removes the last trace property field from the **Trace Properties** dialog.
- **OK**: Applies values and closes the dialog.

- **Apply**: Applies values without closing the dialog.
- **Reset**: Resets all fields to last values applied.
- **Clear**: Clears all fields. Detaches object from data source (once **Apply** or **OK** is selected).
- **Cancel**: Closes the dialog with last values applied.

This property is in the **Trace** property group.

#### traceShadowFlag

Select to enable trace shadows.

This property is in the **Trace** property group.

## Bar graph: X-Axis group

This property group includes a property, xAxisFlag, that controls the visibility of x-axis labels.

#### xAxisFlag

Select to display x-axis labels.

This property is in the **X-Axis** property group.

## Bar graph: Y-Axis group

Properties in this group control the visibility and range of the y-axis or y-axes, as well as y-axis label formats and y-axis divisions. They also control the visibility of y-axis grid lines (but see also "Bar graph: Plot Area group" on page 296).

#### Y-Axis group properties

The group includes the following properties:


- "traceYAxisFlag" on page 300
- "traceYAxisFormat" on page 300
- "traceYAxisMajorDivisions" on page 300
- "traceYAxisMinorDivisions" on page 300
- "yAxisAutoScaleMode" on page 300
- "yAxisFlag" on page 300
- "yAxisFormat" on page 300
- "yAxisGridMode" on page 301
- "yAxisMajorDivisions" on page 301
- "yAxisMinorDivisions" on page 301

#### traceYAxisFlag

Select this property to plot the traces against a y-axis that is separate from the bars. The `traceYAxisFlag` property is unavailable if the drawHorizontalFlag property is selected. The trace y-axis will be drawn to the right of the plot area.

This property is in the **Y-Axis** property group.

#### traceYAxisFormat

Select or enter the numeric format of trace values displayed on the y-axis. Use syntax from the Java `DecimalFormat` class.

This property is in the **Y-Axis** property group.

#### traceYAxisMajorDivisions

Specify the number of major divisions on the trace y-axis. This option only applies if the traceYAxisFlag is on.

This property is in the **Y-Axis** property group.

#### traceYAxisMinorDivisions

Specify the number of minor divisions on the trace y-axis. This option only applies if the traceYAxisFlag is on.

This property is in the **Y-Axis** property group.

#### yAxisAutoScaleMode

Select one of the following modes to control the y-axis range:

- **Off**: The yValueMin and yValueMax properties determine the range of the y-axis. This is the default.
- **On**: The dashboard calculates the y-axis range according to data values being plotted.
- **On - Include Min/Max**: The dashboard calculates the smallest range (with rounding) that includes yValueMin and yValueMax as well as all plotted points.

This property is in the **Y-Axis** property group.

#### yAxisFlag

Select to display the y-axis.

This property is in the **Y-Axis** property group.

#### yAxisFormat

Sets the numeric format of values displayed on the y-axis. Select or enter a format. Use syntax from the Java `DecimalFormat` class.

This property is in the **Y-Axis** property group.

#### yAxisGridMode

Controls the alignment of grid lines drawn to the left and right of the bar graph. Select one of the following:

**Bar Axis**: Align grid lines with the left y-axis. This is the default

**Trace Axis**: Align with the right y-axis.

**Bar and Trace Axis**: Draw two sets of grid lines, one aligned with the left y-axis and the other with the right y-axis.

This property is in the **Y-Axis** property group.

#### yAxisMajorDivisions

Specify the number of major divisions on the y-axis.

This property is in the **Y-Axis** property group.

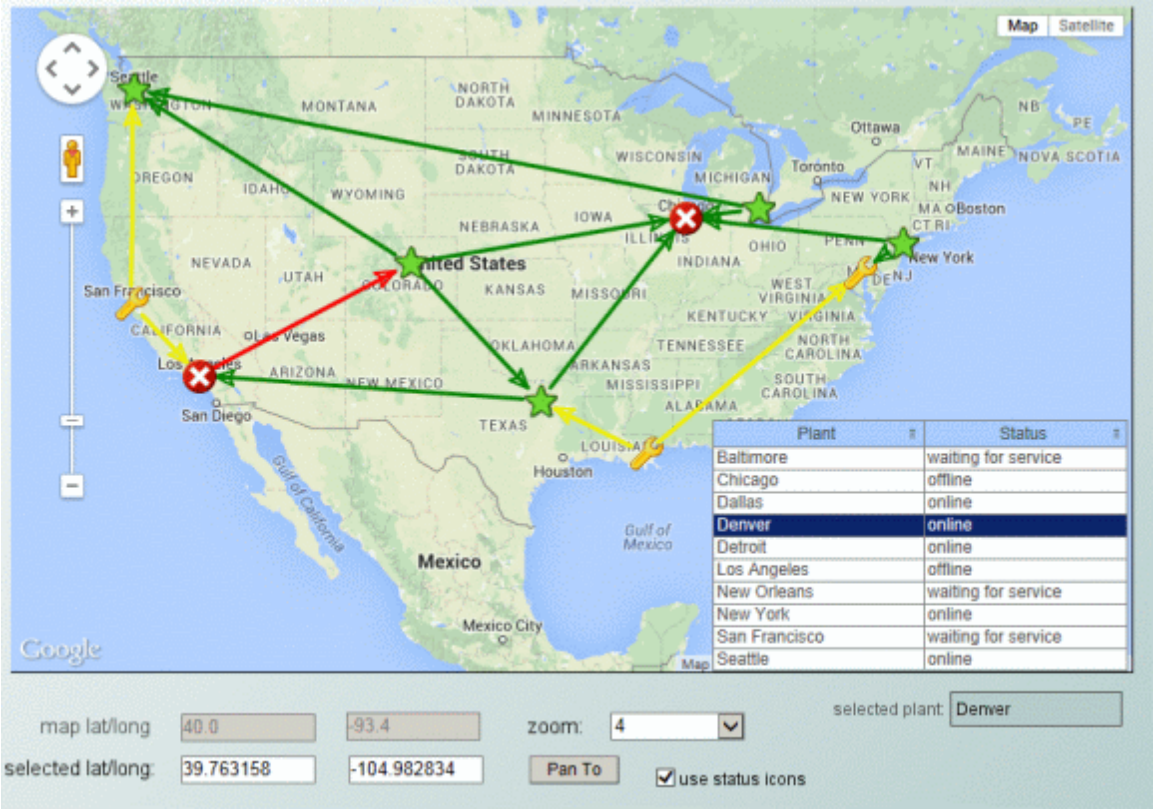
#### yAxisMinorDivisions

Specify the number of minor divisions on the y-axis.

This property is in the **Y-Axis** property group.

## Google map

The map object displays an embedded Google map in Display Server clients.




In the Builder, a Google map instance appears as a gray rectangle which you use to set the size and position of the map object, and also configure the latitude, longitude, zoom level and other map object properties through the Builder's property sheet. The Google map object is rendered only when the display is opened in the Display Server deployment.

In the Builder and Viewer, the Google map object instance appears as an empty gray rectangle. The Viewer does not support the map object.

Using the map object properties and RTView data attachments, the map object can be populated with marker objects at specific latitude/longitude positions, and also links between the markers. RTView drilldown and command operations can be triggered by clicking on the map object, as well as on its markers and links. Double-click, right-click, and `drillDownColumn` substitutions are supported as on other table-driven objects. Operations such as zoom, pan and marker selection can be tied to substitutions.

The Thin Client must have internet access to download the Google Maps Javascript API and map data. The Thin Client supports Google Maps in our supported browsers. The Thin Client loads the Google Maps Javascript API to render and manipulate the map. In most cases, a key or license must be obtained from Google for business use of the Google Maps Javascript API. For details, see the " Licensing" on page 309 section.

The **Object Properties** panel organizes Google map object properties into the groups below.

## Map graph: Data group

The properties in this group specify how data is displayed in the map object. This group contains the following properties:


- "valueTable" on page 303
- " valueTableForLinks" on page 303


### valueTable

To attach data to the map object, right-click **Property Value** field of the `valueTable` property and select **Attach to Data**. The `valueTable` property can be used to place markers (icons) on the map, at specific locations. The property must be attached to a table that contains one row for each marker. The column names are unimportant, but the column order and type must be as follows.

- column 1 (string): The name
- column 2 (number): The latitude
- column 3 (number): The longitude
- column 4 (string): The icon name/path
- column 5 (integer): The icon height in pixels

The first three columns are required, the others are optional. The marker name must be unique as it is used in drilldowns to indicate the selected ("clicked") marker, and can also be used to select a marker.

If column 4 (icon name) is omitted or empty, the default Google Maps marker is used. If column 5 (icon height) exists and its value is greater than zero, the value is used to center the icon on the marker's latitude/longitude position, otherwise the 0,0 pixel of the icon will be placed on the marker's latitude/longitude position.

### valueTableForLinks

Use this property to draw links between markers on the map. The property must be attached to a table that contains one row for each link, with the columns shown below. The column names are unimportant, but the column order and type must be as follows. The first 2 columns are required, the others are optional.

- column 1 (string): The name of marker at start of link
- column 2 (string): The name of marker at end of link
- column 3 (string): The link line color, as a CSS color name or #rrggbb hex value
- column 4 (integer): The link line width
- column 5 (integer): The arrow mode

The first two columns specify the names of the markers at the start and end of the link. These must correspond to the names of markers in the `valueTable`. If column 3 is

omitted, the link color is black. If column 4 is omitted, the link width is 2 pixels. The arrow mode values are 1 (one arrow, pointing to start marker), 2 (one arrow, pointing to end marker) and 3 (two markers, at each end of link). If column 5 is omitted, the default mode of 3 is used.

## Map graph: Interaction group

The properties in this group specify interactions in the map object. This group contains the following properties:


- "command" on page 304
- "commandCloseWindowOnSuccess" on page 304
- "commandConfirm" on page 304
- "commandConfirmText" on page 305
- "drillDownColumnSubs " on page 305
- "drillDownSelectMode " on page 305
- "drillDownTarget " on page 305


### command

Use the `command` property to invoke behavior when the user clicks on the map, a marker, or a link.

### commandCloseWindowOnSuccess

If selected, the window that initiates a system command will automatically close when the system command is executed successfully. This property only applies to system commands. With data source commands, the window is closed whether or not the command is executed successfully.

For multiple commands, this property is applied to each command individually. Therefore if the first command in the multiple command sequence succeeds, the window closes before the rest of the commands are executed.

**Note:** The `commandCloseWindowOnSuccess` property is not supported in Display Server.

### commandConfirm

If selected, the command confirmation dialog is enabled. Use the `commandConfirmText` property to write your own text for the confirmation dialog, otherwise the text from command property will be used.

For multiple commands, if you confirm the execution then all individual commands will be executed in sequence with no further confirmation. If the you cancel the execution, none of the commands in the sequence are executed.

#### commandConfirmText

Enter command confirmation text directly in the **Property Value** field. If `commandConfirmText` is not specified, then the text from command property will be used.

#### drillDownColumnSubs

The `drillDownColumnSubs` property is treated the same as for other table-driven objects. If a marker is clicked, the `drillDownColumnSubs` are set using values from the `valueTable` row that corresponds to that marker. If a link is clicked, the `drillDownColumnSubs` are set using values from the `valueTableForLinks` row that corresponds to that link. In addition, the following predefined substitutions are also set when a drilldown is executed:

- $mapSelLat: The latitude of the map location or marker clicked.
- $mapSelLng: The longitude of the map location or marker clicked.
- $mapLat: The latitude of the map's current center location.
- $mapLng: The longitude of the map's current center location.
- $mapZoom: The current zoom level of the map, a value between 0 and 21.

#### drillDownSelectMode

Control how a drill down display is activated. Select one of the following options:

- **Anywhere**: A click anywhere on the map triggers the object's command or drilldown.
- **Markers**: Only a click on a marker triggers the command or drilldown.
- **Links**: Only a click on a link triggers the command or drilldown.
- **Markers & Links**: Only a click on a marker or a link triggers the command or drilldown. (This is the default setting)

#### drillDownTarget

To specify a drill down display, double click on `drillDownTarget` in the **Property Name** field to bring up the Drill Down Properties dialog. See "Drill-Down Specification" on page 535.

## Map graph: Map group

The properties in this group specify the position, zoom level and label behavior in the map object. This group contains the following properties:


- "labelsZoomThreshold " on page 306
- "latitude" on page 306
- "longitude" on page 306

- "selectedMarker" on page 306
- "zoom" on page 306


#### labelsZoomThreshold

If the `labelsZoomThreshold` property is set to a value greater than zero, a label balloon appears next to each marker when the zoom level is greater than or equal to that value. The label text in the balloon is the name assigned to the marker by the first column in `valueTable`. A label can be closed by clicking its "x" button. The label reappears if the display is closed and reopened or if the zoom threshold is reached. The default value of `labelsZoomThreshold` is zero which disables the labels.

#### latitude

The `latitude` property specifies the latitude of the point on which the map is to be centered.

#### longitude

The `longitude` property specifies the longitude of the point on which the map is to be centered.

#### selectedMarker

The `selectedMarker` property specifies the name of the marker to select.

#### zoom

The `zoom` property specifies the zoom level of the map, between 0 and 21.

## Map graph: Object group

The properties in this group specify the layout in the map object. This group contains the following properties:


- "anchor" on page 306
- "dock" on page 307
- "objHeight" on page 308
- "objName" on page 308
- "objWidth" on page 308
- "objX" on page 308
- "objY" on page 308
- "visFlag" on page 309


#### anchor

Specify where to anchor an object in the display.

**Note:** If an object has the `dock` property set, the `anchor` property will be ignored.

The `anchor` property is applied only when the dimensions of the display are modified, either by editing **Background Properties** or resizing the window in **Layout** mode. Select **None**, or one or more following options:

| | |
|---|---|
| **None** | Object not anchored. This is the default. |
| **Top** | Anchor top of object at top of display. |
| **Left** | Anchor left side of object at left of display. |
| **Bottom** | Anchor bottom of object at bottom of display. |
| **Right** | Anchor right side of object at right of display. |

When a display is resized, the number of pixels between an anchored object and the specified location remain constant. If an object is anchored on opposite sides (that is **Top** and **Bottom** or **Left** and **Right**), the object will be stretched to fill the available space.

#### dock

Specify the docking location of an object in the display. Select from the following options:

| | |
|---|---|
| **None** | Object is not docked. This is the default. |
| **Top** | Dock object at top of display. |
| **Left** | Dock object at left of display. |
| **Bottom** | Dock object at bottom of display. |
| **Right** | Dock object at right of display. |
| **Fill** | Dock object in available space remaining in the display after all docked objects are positioned. |

If the dimensions of the display are modified, either by editing **Background Properties** or resizing the window in **Layout** mode, the properties (`objX`, `objY`, `objWidth` and `objHeight`) of docked objects will automatically adapt to match the new size of the display.

When multiple objects are docked to the same side of the display, the first object is docked against the side of the display, the next object is docked against the edge of the first object, and so on.

When objects are docked to multiple sides of the display, the order in which objects were added to the display controls docking position. For example, the first object added to the display is docked at the **Top** and the second object is docked at the **Left**. Consequently, the first object will fill the entire width of the display and the second object will fill the left side of the display from the bottom of the first object to the bottom of the display.

Objects in a display have the dock property set to **Fill**, are laid out across a grid in the available space remaining after all docked objects are positioned. By default, the grid has one row and as many columns as there are objects in the display. You can modify the grid in the **Background Properties** dialog. Once an object is docked, there are some limitations on how that object can be modified.

- Docked objects cannot be dragged or repositioned using `objX` and `objY` properties.
- Docked objects cannot be resized using the `objWidth` or `objHeight` properties. To resize you must drag on the resize handle.
- Docked objects can only be resized toward the center of the display (for example, if an object is docked at the **Top**, only its height can be increased by dragging down towards the center of the display).
- Docked objects set to **Fill** cannot be resized at all.
- Docked objects cannot be moved using **Align**. Non-docked objects can be aligned against a docked object, but a docked object will not move to align against another object.
- Docked objects are ignored by **Distribute**.

#### objHeight

Set height of the object in pixels.

#### objName

Name given to facilitate object management through the Object List dialog. Select **Tools > Object List**

#### objWidth

Set width of the object in pixels.

#### objX

Set the x-axis position of object.

#### objY

Set the y-axis position of object.

#### visFlag

Controls visibility of the object.

## Licensing

The Google Maps Javascript API requires a key or license from Google. For more information, see: https://developers.google.com/maps/licensing

Apama does not provide a key or license for the Google Maps Javascript API. However, the Thin Client can be configured to download the API using a custom URL that contains specific key or license information obtained from Google.

The custom URL is defined by creating a javascript file named `apama_extra.js` under the *`APAMA_WORK`*`\dashboards` folder. Users can also create the `apama_extra.js` file under the *`dashboards`* folder of a Designer project.

The `apama_extra.js` file contains just one javascript statement. For example,

- to specify a URL containing a Google API key, add the following line:

  ```
  rtv.customGoogleMapsApiBaseURL = "https://maps.googleapis.com/maps/
  api/js?key=YOUR_GOOGLE_API_KEY";
  ```

- to specify a "Google Maps API for Work" client ID and release version 3.20:

  ```
  rtv.customGoogleMapsApiBaseURL = "https://maps.googleapis.com/maps/
  api/js?client=YOUR_GOOGLE_CLIENT_ID&v=3.20";
  ```

If `apama_extra.js` is found under the *`dashboards`* folder of the Designer project, then this javascript file will be used for the Display Server deployment. Otherwise, the system will try to look for `apama_extra.js` under the *`APAMA_WORK`*`\dashboards` folder.

If no `apama_extra.js` is found, the Thin Client uses the following public URL to download the latest "experimental" version of the API with no key or license information: https://maps.googleapis.com/maps/api/js.

**Important:** Please refer to : https://developers.google.com/maps/licensing for information about the license key in order to use this Google Map object.

## Heat map

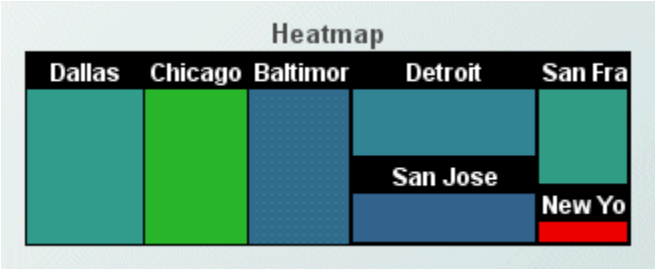
Heat maps visualize data by displaying rectangles of various sizes and colors. Complex heat maps display a hierarchy of rectangles, where a rectangle's level in the hierarchy is represented by its level of geometric nesting within other rectangles.

Heat maps visualize tabular data that contains one or more *index* columns as well as one or more numerical columns.

You specify the data to be visualized with the `valueTable` property.

You designate the index columns by specifying a column name or list of column names as the value of the property `nodeIndexColumnNames`.

If there are two or more non-index numerical columns, the first is the *size-data* column and the second is the *color-data* column. If there is only one non-index numerical column, it serves as both the size-data column and the color-data column.




## Heat maps with one index column

A heat map with a single index column contains one rectangle for each unique value in the index column.

For a given index value, the area of the corresponding rectangle is proportional to the result of aggregating the values in the size-data column in those rows whose index column contains the index value. You specify the type of aggregation to use (`sum`, `count`, `average`, `min`, or `max`) with the `sizeValueGroupType` property.

In addition, for a given index value, the color of the corresponding rectangle is determined by the result of aggregating the values in the color-data column in those rows whose index column contains the index value (see "Mapping from possible aggregation results to colors" on page 311). You specify the type of aggregation to use (`sum`, `count`, `average`, `min`, or `max`) with the `colorValueGroupType` property.

**Important:** Negative aggregated values are treated as **0**.

## Heat maps with multiple index columns

Heat maps with multiple index columns display a rectangle hierarchy. The number of levels of the hierarchy is the number of columns from the visualized data table that are specified as index columns.

In such a heat map, there is a rectangle at level *n* for each unique sequence of values from the first *n* index columns, for every level between 1 and the number of index columns, inclusive.

For a given such sequence of *n* index values, the area of the corresponding rectangle is proportional to the result of aggregating the values in the size-data column in those rows whose first *n* index columns contain the values in the sequence. You specify the type of aggregation to use (`sum`, `count`, `average`, `min`, or `max`) with the `sizeValueGroupType` property.

In addition, for a given such sequence of *n* index values, the color of the corresponding rectangle is determined by the result of aggregating the values in the color-data column in those rows whose first *n* index columns contain the values in the sequence. You specify the type of aggregation to use (`sum`, `count`, `average`, `min`, or `max`) with the `colorValueGroupType` property.

**Important:** Negative aggregated values are treated as **0**.

## Mapping from possible aggregation results to colors

The possible color-data aggregation results are mapped to colors as follows:

- If `colorValueAutoScaleMode` is `Off`
  - The possible aggregation result value specified in `colorValueMin` is mapped to the color specified by `minColor`.
  - The possible aggregation result value specified in `colorValueMax` is mapped to the color specified by `maxColor`.
- If `colorValueAutoScaleMode` is `On`
  - The smallest actual aggregation result for the current display is mapped to the color specified by `minColor`.
  - The largest actual aggregation result for the current display is mapped to the color specified by `maxColor`.
- If `colorValueAutoScaleMode` is `Off - Include Min/Max`
  - `minColor` is mapped to the smaller of `colorValueMin` and the smallest actual aggregation result for the current display.
  - `maxColor` is larger of `colorValueMax` and the largest actual aggregation result for the current display.

In all three cases, possible aggregation result values that are in between those mapped to `minColor` and `maxColor` are mapped through interpolation, using the colors between `minColor` and `maxColor` arranged either in gradient order or color-wheel order (as determined by `linearColorMappingFlag`).

## Drill down displays

Since data in a heat map is aggregated, the value shown in a node might not be the same as the value passed down to a drill down display. For example, suppose your heat map is attached to a table where the index column is **Plant** and the size column is **Units Completed**. If you have two rows where the **Plant** is **San Francisco**, then the node size is based on the total of the **Units Completed** values for both rows. However when you drill down, the drill down value for **Units Completed** will be the value in the first row in the table where the **Plant** is **San Francisco**.

## Object class name

When a heat map object is selected in the Builder canvas, the **Object Class Name** that appears at the top of the **Object Properties** pane is **obj_heatmap**.

## Heat map property groups

The **Object Properties** panel organizes heat map properties into the following groups:


- "Heat map: Background properties" on page 312
- "Heat map: Data group" on page 315
- "Heat map: Data format group" on page 317
- "Heat map: Data Label group" on page 318
- "Heat map: Historian group" on page 318
- "Heat map: Interaction group" on page 318
- "Heat map: Label group" on page 321
- "Heat map: Layout group" on page 323
- "Heat map: Node group" on page 324
- "Heat map: Object group" on page 325
- "Heat map: Quality group" on page 327


## Heat map: Background properties

Properties in this group control the visibility and appearance of the map's background.

### Background properties

The group contains the following properties:


- "bgBorderColor" on page 313
- "bgBorderFlag" on page 313
- "bgColor " on page 313
- "bgEdgeWidth " on page 313
- "bgGradientColor2" on page 313
- "bgGradientMode" on page 313
- "bgRaisedFlag" on page 314
- "bgRoundness" on page 314
- "bgShadowFlag" on page 314
- "bgStyle" on page 314

- "bgVisFlag " on page 314
- "borderPixels " on page 315


#### bgBorderColor

Sets the color of the border (see `bgBorderFlag`) of the background rectangle. Select the **...** button and choose a color from the palette. Close the Color Chooser window when you are done.

#### bgBorderFlag

Select to display a border around the background rectangle.

This property is in the **Background** property group.

#### bgColor

Sets the background color. Select the ... button and choose a color from the palette. Close the Color Chooser window when you are done.

This property is in the **Background** property group.

#### bgEdgeWidth

Sets the width in pixels of the 3D edge on the background rectangle. This property is only used if `bgBorderFlag` is selected.

This property is in the **Background** property group.

#### bgGradientColor2

Sets the color for the second color in the gradient. The default is white. The `bgColor` property sets the first color in the gradient.

This property is in the **Background** property group.

#### bgGradientMode

Display a gradient in the background rectangle. Select from the following options:

- **None**: No gradient.
- **Diagonal Edge**: Gradient is drawn at a 45 degree angle from the top left to the bottom right corner of the object.
- **Diagonal Center**: Gradient is drawn at a 45 degree angle from the center to the top left and the bottom right corners of the object.
- **Horizontal Edge**: Gradient is drawn horizontally from the top to the bottom of the object.
- **Horizontal Center**: Gradient is drawn horizontally from the center to the top and bottom of the object.

- **Vertical Edge**: Gradient is drawn vertically from the left to the right of the object.
- **Vertical Center**: Gradient is drawn vertically from the center to the left and right of the object.

This property is in the **Background** property group.

#### bgRaisedFlag

Reverses the direction of the gradient, as well as that of the 3D edge if the `bgStyle` selected is 3D Rectangle.

This property is in the **Background** property group.

#### bgRoundness

Sets the arc length of the rounded corners. This property is only available if the `bgStyle` selected is **Round Rectangle**.

The value of `bgRoundness` cannot exceed half the value of the `objWidth` or the `objHeight`. If `bgRoundness` does exceed that value, half of `objWidth` or `objHeight` (whichever is smaller) will be used instead. For example if `objWidth` is `100` and `objHeight` is `50`, then the value of `bgRoundness` cannot exceed `25`. If it does, then half the value of `objHeight` (`25`) will be used instead. This property is in the **Background** property group.

#### bgShadowFlag

Select to display a drop shadow on the background rectangle.

This property is in the **Background** property group.

#### bgStyle

Choose one of the following three options from the drop down menu:

- **Rectangle**: Select to display a background rectangle.
- **3D Rectangle**: Select to display a 3D edge on the background rectangle. If selected, use `bgEdgeWidth` to set the width of the 3D edge.
- **Round Rectangle**: Select to display a background rectangle with rounded edges. If selected, use `bgRoundness` to set the arc length of the rounded corners.

This property is in the **Background** property group.

#### bgVisFlag

Select to display the background rectangle.

This property is in the **Background** property group.

#### borderPixels

Sets the width in pixels of the border between the chart and the edge of the background rectangle.

This property is in the **Background** property group.

## Heat map: Data group

Properties in this group control what data appears in the heat map, and how it is mapped to node size and color.

#### Data group properties

The group contains the following properties:

- "colorValueAutoScaleMode" on page 315
- "colorValueDivisor" on page 315
- "colorValueGroupType" on page 315
- "colorValueMax" on page 316
- "colorValueMin" on page 316
- "nodeIndexColumnNames" on page 316
- "sizeValueDivisor" on page 316
- "sizeValueGroupType" on page 316
- "valueTable" on page 281

#### colorValueAutoScaleMode

Controls how aggregation results are mapped to colors. See "Mapping from possible aggregation results to colors" on page 311.

This property is in the **Data** property group.

#### colorValueDivisor

Divides `colorValueMin`, `colorValueMax`, and color-data aggregation results by the specified value.

This property is in the **Data** property group.

#### colorValueGroupType

Sets the type of aggregation to use for color data: **sum**, **average**, **count**, **min**, or **max**.

This property is in the **Data** property group.

#### colorValueMax

Controls how aggregation results are mapped to colors. See "Mapping from possible aggregation results to colors" on page 311.

This property is in the **Data** property group.

#### colorValueMin

Controls how aggregation results are mapped to colors. See "Mapping from possible aggregation results to colors" on page 311.

This property is in the **Data** property group.

#### nodeIndexColumnNames

Specify a semicolon-delimited list of index column names. If not specified, the first text column in the table attached to `valueTable` is used as the index column and the first two numeric columns are used as data columns.

This property is in the **Data** property group.

#### sizeValueDivisor

sizeValueDivisor Divides size-data aggregation results by the specified value.

.This property is in the **Data** property group.

#### sizeValueGroupType

Sets the type of aggregation to use for size data: **sum**, **average**, **count**, **min**, or **max**.

This property is in the **Data** property group.

#### valueTable

Specifies the data to be visualized. Tabular data attached to the `valueTable` property must contain one or more index columns and at least one data column. The heat map displays one level of nodes for each index column specified. Use the `nodeIndexColumnNames` property to specify column names. The first non-index numeric data column is used to control the size of each node. The second non-index numeric data column is used to control the color of the node. If only one data column is specified, it controls both node size and node color.

Data attached to `valueTable` is aggregated by unique index value. *Note:* Negative aggregated values are treated as 0. By default, both size and color data is subtotaled. Alternately, you can specify aggregation types using the `colorValueGroupType` and `sizeValueGroupType` properties.

See "Heat map" on page 309 for more information.

This property is in the **Data** property group.

## Heat map: Data format group

Properties in this group control the format of tooltip-displayed data, as well as the mapping from color data to colors.

### Data group properties

The group contains the following properties:

- "colorValueFormat" on page 317
- "linearColorMappingFlag" on page 317
- "maxColor" on page 317
- "minColor" on page 317
- "sizeValueFormat" on page 317

### colorValueFormat

Sets the numeric format of the color value displayed in tooltips. Use syntax from the Java `DecimalFormat` class. To enable tooltips, select the `mouseOverFlag`.

This property is in the **Data Format** property group.

### linearColorMappingFlag

If selected, possible aggregation result values that are in between those mapped to `minColor` and `maxColor` are mapped through interpolation, using the colors between `minColor` and `maxColor` arranged in gradient order. If deselected, the interpolation uses the colors arranged in color-wheel order.

This property is in the **Data Format** property group.

### maxColor

Sets the maximum color. Possible node colors range from the `minColor` to `maxColor`. See "Mapping from possible aggregation results to colors" on page 311.

This property is in the **Data Format** property group.

### minColor

Sets the minimum color. Possible node colors range from the `minColor` to `maxColor`. See "Mapping from possible aggregation results to colors" on page 311.

This property is in the **Data Format** property group.

### sizeValueFormat

Sets the numeric format of the size value displayed in tooltips. Use syntax from the Java `DecimalFormat` class. To enable tooltips, select the `mouseOverFlag`.

This property is in the **Data Format** property group.

## Heat map: Data Label group

The property in this group, columnDisplayNames, sets alternate display names for column names.

#### columnDisplayNames

Sets alternate display names for column names in your heat map data. Column names are displayed in tooltips.

This property is in the **Data Label** property group.

## Heat map: Historian group

Do not use the properties in this group.

#### historyTableName

Do not use this property.

This property is in the **Historian** property group.

#### historyTableRowNameFlag

Do not use this property.

This property is in the **Historian** property group.

## Heat map: Interaction group

Properties in this group control various forms of interaction between the end user and the graph, including scrolling, highlighting, and activating commands, drill downs, and tooltips.

#### Interaction group properties

The group includes the following properties:


- "command" on page 319
- "commandCloseWindowOnSuccess" on page 319
- "commandConfirm" on page 319
- "commandConfirmText" on page 320
- "drillDownColumnSubs" on page 320
- "drillDownSelectMode" on page 320
- "drillDownTarget" on page 321
- "mouseOverAdditionalColumns" on page 321
- "mouseOverDefaultColumnsFlag" on page 321

- "mouseOverFlag" on page 321

#### command

Assign a command or group of commands to this stock chart by right-clicking on the `command` property name in the Object Properties window. Select **Define Command** and choose **SYSTEM**, **APAMA**, or **MULTIPLE**. For more information, see "Using the Define Apama Command dialog" on page 226.

Once a command or command group has been assigned to this object, you can activate it from a deployed dashboard or from the Dashboard Builder:

- Dashboard Builder: Double click on the object.
- Web-based deployment: Single click on the object or else right click on it and select **Execute Command** from the popup menu.
- Local deployment: By default, single-click on the object or else right-click on it and select **Execute Command** from the popup menu. To override the default, select **Tools > Options** in the Builder (do this before you generate the deployment package), and uncheck **Single-Click for Drill Down and Commands** in the **General** tab. This allows the end user to use either a double click or a right click.

When you activate a command, any defined drill down substitutions are performed, and then the command is executed.

If you assign multiple commands, the commands are launched in an arbitrary order, and are executed asynchronously; there is no guarantee that one command will finish before the next one in the sequence starts.

This property is in the **Interaction** property group.

#### commandCloseWindowOnSuccess

Select this property to automatically close the window that initiates a **SYSTEM** command when the command is executed successfully. This applies to **SYSTEM** commands only, and is not supported at all for thin-client, Web-page deployments.

With **APAMA** commands, the window is closed whether or not the command is executed successfully. For **MULTIPLE** commands, the window closes when the first command in the command group succeeds.

This property is in the **Interaction** property group.

#### commandConfirm

By default, when the end user executes a command (see the `command` property), the command confirmation dialog is disabled. To control this option for each individual object, use the `commandConfirm` check box. If confirmation is required for a **MULTIPLE** command group, a single confirmation dialog is presented; if you confirm the execution, all individual commands in the group are executed with no further confirmation. If the you cancel the execution, none of the commands in the group is executed.

You can also override the confirmation status of individual objects with an application-wide policy. Select **Tools | Options** and choose from three confirmation values:

- **Do not confirm**: Indicates that no commands require confirmation (regardless of each object's confirmation status).
- **Confirm all**: Indicates that all commands require confirmation (regardless of each object's confirmation status).
- **Use object confirm flag** (default): Indicates that the confirmation status of each object will determine whether confirmation is required.

This property is in the **Interaction** property group.

#### commandConfirmText

Use this property to write your own text for the confirmation dialog. Otherwise, default text is used. See `commandConfirm`.

This property is in the **Interaction** property group.

#### drillDownColumnSubs

Use this property to direct a dashboard to assign data-table column values to specified dashboard variables when the end user activates a drilldown on this object. In the **Object Properties** window, double-click on `drillDownColumnSubs` in the **Property Name** field to bring up the **Drill Down Column Substitutions** dialog.

The dialog has the following fields and buttons:

- **Substitution String**: Enter the dashboard variable next to the name of the data table column whose value you want assigned to the variable. Press **Enter**.
- **Add Column**: Enter the name of a column and click the **Add Column** button to insert a column into the table.
- **Clear**: Click the **Clear** button to remove all variables listed.

The **Column Name** list is populated based on the table's data attachment. If you have not yet attached the table to data, this list is empty.

Once you have selected which column values to pass in as substitutions, double-click on any element in your object to open a drill down window that displays corresponding values.

This property is in the **Interaction** property group.

#### drillDownSelectMode

Use this property to control how a drill down display is activated. Select one of the following:

- **Anywhere** to activate a drill down display by double-clicking anywhere on the chart.

- **Element Only** to enable a drill down display only when you double-click on an element of the chart, such as a bar or candlestick.

This property is in the **Interaction** property group.

#### drillDownTarget

To specify a drill down display, double click on `drillDownTarget` in the **Property Name** field to bring up the Drill Down Properties dialog. See "Drill-Down Specification" on page 535.

This property is in the **Interaction** property group.

#### mouseOverAdditionalColumns

Select the button to open a dialog to select which columns to include in tooltips and, optionally, specify a date format (or other numeric format) and value divisor (for numeric columns) for each column displayed. In the tool tip, the name and value for each selected column is displayed. If the `mouseOverDefaultColumnsFlag` is selected, then columns you include are inserted following the default columns in the tooltip. If specified, `columnDisplayNames` are applied to the columns you selected to include.

This property is in the **Interaction** property group.

#### mouseOverDefaultColumnsFlag

Select to include column names and values from `valueTable` (for index columns and data columns) in tooltips. If `columnDisplayNames` are specified, they will be applied to all column names.

This property is in the **Interaction** property group.

#### mouseOverFlag

Select to enable tooltips for your heat map. To display a tooltip, select the map and point to a node with your mouse. The tooltip will contain information from your data attachment about that node.

> **Note:** Heat maps containing large data sets may run slowly on the Display Server if `mouseOverFlag` is selected.

This property is in the **Interaction** property group.

## Heat map: Label group

Properties in this group control the graph's main label (which defaults to **Heatmap**), including text, alignment, color, font, and size.

#### Label group properties

The group includes the following properties:

- "label" on page 322
- "labelMinTabWidth" on page 322
- "labelTextAlignX" on page 322
- "labelTextAlignY" on page 322
- "labelTextColor" on page 323
- "labelTextFont" on page 323
- "labelTextHeight" on page 323


#### label

Specifies the text for the chart label. Click the ellipsis for multi-line text.

The default is **Heatmap**.

This property is in the **Label** property group.

#### labelMinTabWidth

Sets minimum width of the label tab. This property only applies if `labelTextAlignY` is set to **TabTop**.

This property is in the **Label** property group.

#### labelTextAlignX

Sets the x-axis alignment of the chart label (see the `label` property). Select **Left**, **Center**, or **Right** from the drop down list.

This property is in the **Label** property group.

#### labelTextAlignY

Sets the y-axis position of the chart label (see the `label` property). Select one of the following from the drop down list:

- **Outside Top**: Well above the background rectangle
- **Top**: Just above the background rectangle
- **Title Top**: Along the top line of the background rectangle
- **Tab Top**: Just above the background rectangle. Height and width of the tab is dependent on the height and width of the text. Use the `labelMinTabWidth` property to specify a minimum tab width.
- **Inside Top**: Inside the top of the background rectangle

This property is in the **Label** property group.

#### labelTextColor

Specifies the color of the chart label text (see the `label` property). Select the **...** button and choose a color from the palette. Close the Color Chooser window when you are done.

This property is in the **Label** property group.

#### labelTextFont

Specifies the font of the chart label text (see the `label` property). Select an item from drop down list.

This property is in the **Label** property group.

#### labelTextHeight

Specifies the point size of the chart label text (see the `label` property).

This property is in the **Label** property group.

## Heat map: Layout group

Properties in this group affect the layout of nodes in the heat map.

#### Layout group properties

This group contains the following properties:

- "adjustSizeForLabelFlag" on page 323
- "layoutStyle" on page 323

#### adjustSizeForLabelFlag

Select to compress the ratio between the smaller nodes and larger nodes so that the size of smaller nodes is increased to accommodate labels.

> **Note:** This property only applies to nodes that display labels.

This property is in the **Layout** property group.

#### layoutStyle

Select from the following layout styles:

- **Squarified**: Nodes are more square in shape and ordered according to the size of the value from the top-left to the bottom-right.
- **Strip**: Nodes are more square in shape and ordered according to the order of the rows in the `valueTable`.
- **Slice Horizontal**: Nodes are short and wide and ordered according to the order of the rows in the `valueTable`.

- **Slice Vertical**: Nodes are tall and narrow and ordered according to the order of the rows in the `valueTable`.
- **Slice Best**: Nodes are laid out either like **Slice Horizontal** or **Slice Vertical** based on what fits best in the available space.
- **Slice Alternate Horizontal**: The layout alternates between **Slice Horizontal** and **Slice Vertical** based on the node depth. The top level nodes use **Slice Horizontal**.
- **Slice Alternate Vertical**: The layout alternates between **Slice Horizontal and Slice Vertical** based on the node depth. The top level nodes use **Slice Vertical**.

This property is in the **Layout** property group.

## Heat map: Node group

Properties in this group affect the appearance of nodes in the heat map.

### Node group properties

This group contains the following properties:


- "nodeBgBorderHighlightFlag" on page 324
- "nodeBgBorderSize" on page 324
- "nodeBgColor" on page 325
- "nodeLabelNestDepth" on page 325
- "nodeLabelTextColor" on page 325
- "nodeLabelTextFont" on page 325
- "nodeLabelTextHeight" on page 325
- "nodeLabelVisFlag" on page 325


#### nodeBgBorderHighlightFlag

Select to draw a border highlight around the nodes.

> **Note:** This property is ignored if the `nodeBgBorderSize` is set to 0 or 1.

This property is in the **Node** property group.

#### nodeBgBorderSize

Specify (in pixels) the size of the border between nodes. If set to -1, the deepest nested level of nodes has a one pixel border and the border increases by two pixels for each level of nesting.

This property is in the **Node** property group.

#### nodeBgColor

Select the button and choose from the palette to set the background color for the nodes.

This property is in the **Node** property group.

#### nodeLabelNestDepth

Specify the number of nest levels to display node labels. If set to 0, then no labels are displayed.

This property is in the **Node** property group.

#### nodeLabelTextColor

Select the button and choose from the palette to set the text color for the node labels.

This property is in the **Node** property group.

#### nodeLabelTextFont

Select the font to use for the node labels.

This property is in the **Node** property group.

#### nodeLabelTextHeight

Specify the text height for the node labels.

This property is in the **Node** property group.

#### nodeLabelVisFlag

Select to display labels on the nodes.

> **Note:** This property is ignored if `nodeLabelNestDepth` is set to 0.

This property is in the **Node** property group.

## Heat map: Object group

Properties in this group control the visibility and transparency of the heat map as a whole. They also control (or reflect) the overall position and dimensions of the heat map. In addition, a property in this group reflects the generated name of this individual heat map.

#### Object group properties

This group contains the following properties:

- "anchor" on page 326
- "dock" on page 326

- "objHeight" on page 326
- "objName" on page 326
- "objWidth" on page 326
- "objX" on page 327
- "objY" on page 327
- "transparencyPercent" on page 327
- "visFlag" on page 327


#### anchor

Select zero or more of **Top**, **Left**, **Bottom**, and **Right** in order to control the object's placement. The `anchor` property is only applied when the display is resized either by changing the **Background Properties** on the display or by resizing the window in **Layout** mode. If an object has the `dock` property set, the `anchor` property is ignored. See "About resize modes" on page 39.

#### dock

Select **None** (default), **Top**, **Left**, **Bottom**, **Right**, or **Fill** in order to control the object's placement in **Layout** resize mode. See "About resize modes" on page 39.

#### objHeight

Set the height of a chart by entering a value for this property or by dragging a handle of the bounding box that appears when the chart is selected. When you drag a handle of the bounding box, the displayed value for this property changes to reflect the real-time height of the chart.

This property is in the **Object** property group.

#### objName

An identifier that is generated by the Dashboard Builder. This name can be used by other objects' properties in order to refer to the named chart, as with, for example, the graphName property of the Legend visualization object.

This property is in the **Object** property group.

#### objWidth

Set the width of a chart by entering a value for this property or by dragging a handle of the bounding box that appears when the chart is selected. When you drag a handle of the bounding box, the displayed value for this property changes to reflect the real-time width of the chart.

This property is in the **Object** property group.

#### objX

Sets the X coordinate of the center of this visualization object, relative to the lower left corner of the current dashboard. This value is set automatically when you position the object with the mouse.

This property is in the **Object** property group.

#### objY

Sets the Y coordinate of the center of this visualization object, relative to the lower left corner of the current dashboard. This value is set automatically when you position the object with the mouse.

This property is in the **Object** property group.

#### transparencyPercent

Sets the transparency of this chart.

This property is in the **Object** property group.

#### visFlag

Deselect to make this visualization object invisible in the current dashboard.

This property is in the **Object** property group.

## Heat map: Quality group

Properties in this group allow you to color nodes based on data quality.

#### Quality group properties

This group contains the following properties:

- "valueQuality" on page 327
- "valueQualityColumnName" on page 328
- "valueQualityEnabledFlag" on page 328
- "valueQualityLostData" on page 328
- "valueQualityLostDataColor" on page 328
- "valueQualityNoData" on page 328
- "valueQualityNoDataColor" on page 328

#### valueQuality

Specify a value to compare to settings for the `valueQualityLostData` and `valueQualityNoData` properties. If the specified `valueQuality` matches, the selected corresponding `valueQuality*Color` is applied to all nodes in the heat map.

**Note:** The `valueQuality` property is ignored if the `valueQualityEnabledFlag` is deselected.

This property is in the **Quality** property group.

#### valueQualityColumnName

Specify a column in the `valueTable` to compare, per row, to settings for the `valueQualityLostData` and `valueQualityNoData` properties. If values in the specified `valueQualityColumnName` match, the selected corresponding `valueQuality*Color` is selectively applied to each node in the heat map. If the `valueTable` contains multiple rows for a single index, the highest data quality value is used.

**Note:** The `valueQualityColumnName` property is ignored if the `valueQualityEnabledFlag` is deselected.

This property is in the **Quality** property group.

#### valueQualityEnabledFlag

If selected, nodes are colored based on data quality.

This property is in the **Quality** property group.

#### valueQualityLostData

Enter the lost data value.

This property is in the **Quality** property group.

#### valueQualityLostDataColor

Select the button and choose from the palette to set the node color if the value matches the specified `valueQualityLostData`.

This property is in the **Quality** property group.

#### valueQualityNoData

Enter the no data value.

This property is in the **Quality** property group.

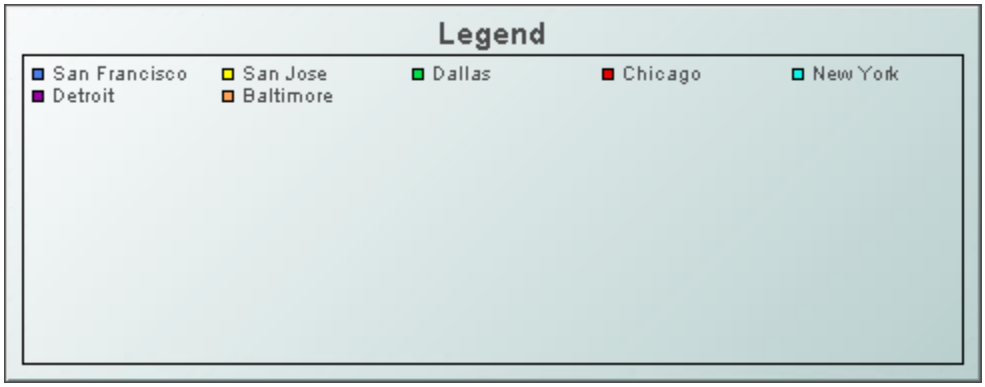
#### valueQualityNoDataColor

Select the button and choose from the palette to set the node color if the value matches the specified `valueQualityNoData`.

This property is in the **Quality** property group.

# Legend

The legend visualization object is useful for displaying a legend that is too lengthy for the built-in legends of the graph objects.




You can use a legend visualization object in conjunction with a bar graph, pie graph, radar graph, or XY graph.

The legend displays information from the graph object to which it is *connected*. Connect a legend to a graph object by setting the legend's graphName property to the value of the graph's `objName` property. Set up all formatting for the legend data in the graph object that it will reflect.

When this visualization object is selected in the Builder canvas, the **Object Class Name** that appears at the top of the **Object Properties** pane is **obj_legend**.

The **Object Properties** panel organizes legend properties into the groups below.

## Legend: Background group

Properties in this group control the visibility and appearance of the legend's outer rectangle, which serves as the background of both the label (see "label" on page 334) and the legend's inner rectangle (see "Legend: Legend group" on page 335).

### Background group properties

The group contains the following properties:

- "bgBorderColor" on page 330
- "bgBorderFlag" on page 330
- "bgColor" on page 330
- "bgEdgeWidth" on page 330
- "bgGradientColor2" on page 330
- "bgGradientMode" on page 330
- "bgRaisedFlag" on page 331

- "bgRoundness" on page 331
- "bgShadowFlag" on page 331
- "bgStyleFlag" on page 331
- "bgVisFlag" on page 331
- "borderPixels" on page 332


#### bgBorderColor

Sets the color of the border (see `bgBorderFlag`) of the background rectangle. Select the **...** button and choose a color from the palette. Close the Color Chooser window when you are done.

#### bgBorderFlag

Select to display a border around the background rectangle.

This property is in the **Background** property group.

#### bgColor

Sets the background color. Select the **...** button and choose a color from the palette. Close the Color Chooser window when you are done.

This property is in the **Background** property group.

#### bgEdgeWidth

Sets the width in pixels of the 3D edge on the background rectangle. This property is only used if `bgBorderFlag` is selected.

This property is in the **Background** property group.

#### bgGradientColor2

Sets the color for the second color in the gradient. The default is white. The `bgColor` property sets the first color in the gradient.

This property is in the **Background** property group.

#### bgGradientMode

Display a gradient in the background rectangle. Select from the following options:

- **None**: No gradient.
- **Diagonal Edge**: Gradient is drawn at a 45 degree angle from the top left to the bottom right corner of the object.
- **Diagonal Center**: Gradient is drawn at a 45 degree angle from the center to the top left and the bottom right corners of the object.

- **Horizontal Edge**: Gradient is drawn horizontally from the top to the bottom of the object.
- **Horizontal Center**: Gradient is drawn horizontally from the center to the top and bottom of the object.
- **Vertical Edge**: Gradient is drawn vertically from the left to the right of the object.
- **Vertical Center**: Gradient is drawn vertically from the center to the left and right of the object.

This property is in the **Background** property group.

#### bgRaisedFlag

Reverses the direction of the gradient, as well as that of the 3D edge if the `bgStyle` selected is 3D Rectangle.

This property is in the **Background** property group.

#### bgRoundness

Sets the arc length of the rounded corners. This property is only available if the `bgStyle` selected is **Round Rectangle**.

The value of `bgRoundness` cannot exceed half the value of the `objWidth` or the `objHeight`. If `bgRoundness` does exceed that value, half of `objWidth` or `objHeight` (whichever is smaller) will be used instead. For example if `objWidth` is `100` and `objHeight` is `50`, then the value of `bgRoundness` cannot exceed `25`. If it does, then half the value of `objHeight` (`25`) will be used instead. This property is in the **Background** property group.

#### bgShadowFlag

Select to display a drop shadow on the background rectangle.

This property is in the **Background** property group.

#### bgStyleFlag

Choose one of the following three options from the drop down menu:

- **Rectangle**: Select to display a background rectangle.
- **3D Rectangle**: Select to display a 3D edge on the background rectangle. If selected, use `bgEdgeWidth` to set the width of the 3D edge.
- **Round Rectangle**: Select to display a background rectangle with rounded edges. If selected, use `bgRoundness` to set the arc length of the rounded corners.

This property is in the **Background** property group.

#### bgVisFlag

Select to display the background rectangle.

This property is in the **Background** property group.

#### borderPixels

Sets the width in pixels of the border between the chart and the edge of the background rectangle.

This property is in the **Background** property group.

## Legend: Data group

The property in this group, graphName, controls what data appears in the graph.

#### graphName

To attach your legend to a given graph object, set this property to the value `objName` for the given graph object.

This property is in the **Data** property group.

## Legend: Historian group

Do not use the properties in this group.

#### historyTableName

Do not use this property.

This property is in the **Historian** property group.

#### historyTableRowNameFlag

Do not use this property.

This property is in the **Historian** property group.

## Legend: Interaction group

Properties in this group configure interaction between the end user and the graph, including commands and drill down interactions.

#### Interaction group properties

The group includes the following properties:


- "command" on page 333
- "commandCloseWindowOnSuccess" on page 333
- "commandConfirm" on page 333
- "confirmText" on page 334
- "drillDownTarget" on page 334

#### command

Assign a command or group of commands to this stock chart by right-clicking on the `command` property name in the Object Properties window. Select **Define Command** and choose **SYSTEM**, **APAMA**, or **MULTIPLE**. See "Using the Define Apama Command dialog" on page 226.

Once a command or command group has been assigned to this object, you can activate it from a deployed dashboard or from the Dashboard Builder:

- Dashboard Builder: Double click on the object.
- Web-based deployment: Single click on the object or else right click on it and select **Execute Command** from the popup menu.
- Local deployment: By default, single-click on the object or else right-click on it and select **Execute Command** from the popup menu. To override the default, select **Tools > Options** in the Builder (do this before you generate the deployment package), and uncheck **Single-Click for Drill Down and Commands** in the **General** tab. This allows the end user to use either a double click or a right click.

When you activate a command, any defined drill down substitutions are performed, and then the command is executed.

If you assign multiple commands, the commands are launched in an arbitrary order, and are executed asynchronously; there is no guarantee that one command will finish before the next one in the sequence starts.

This property is in the **Interaction** property group.

#### commandCloseWindowOnSuccess

Select this property to automatically close the window that initiates a **SYSTEM** command when the command is executed successfully. This applies to **SYSTEM** commands only, and is not supported at all for thin-client, Web-page deployments.

With **APAMA** commands, the window is closed whether or not the command is executed successfully. For **MULTIPLE** commands, the window closes when the first command in the command group succeeds.

This property is in the **Interaction** property group.

#### commandConfirm

By default, when the end user executes a command (see the `command` property), the command confirmation dialog is disabled. To control this option for each individual object, use the `commandConfirm` check box. If confirmation is required for a **MULTIPLE** command group, a single confirmation dialog is presented; if you confirm the execution, all individual commands in the group are executed with no further confirmation. If the you cancel the execution, none of the commands in the group is executed.

You can also override the confirmation status of individual objects with an application-wide policy. Select **Tools | Options** and choose from three confirmation values:

- **Do not confirm**: Indicates that no commands require confirmation (regardless of each object's confirmation status).
- **Confirm all**: Indicates that all commands require confirmation (regardless of each object's confirmation status).
- **Use object confirm flag** (default): Indicates that the confirmation status of each object will determine whether confirmation is required.

This property is in the **Interaction** property group.

#### confirmText

Use this property to write your own text for the confirmation dialog. Otherwise, default text is used. See `commandConfirm`.

This property is in the **Interaction** property group.

#### drillDownTarget

To specify a drill down display, double click on `drillDownTarget` in the **Property Name** field to bring up the Drill Down Properties dialog. See "Drill-Down Specification" on page 535.

This property is in the **Interaction** property group.

## Legend: Label group

Properties in this group control the legend's main label (which defaults to **Legend**), including text, alignment, color, font, and size.

#### Label group properties

The group includes the following properties:

- "label" on page 334
- "labelTextAlignX" on page 335
- "labelTextColor" on page 335
- "labelTextFont" on page 335
- "labelTextHeight" on page 335

#### label

Specifies the text for the chart label. Click the ellipsis for multi-line text.

The default is **Legend**.

This property is in the **Label** property group.

#### labelTextAlignX

Sets the alignment of the chart label (see the label property). Select **Left**, **Center**, or **Right** from the drop down list.

This property is in the **Label** property group.

#### labelTextColor

Specifies the color of the chart label text (see the label property). Select the ... button and choose a color from the palette. Close the **Color Chooser** window when you are done.

This property is in the **Label** property group.

#### labelTextFont

Specifies the font of the chart label text (see the label property). Select an item from drop down list.

This property is in the **Label** property group.

#### labelTextHeight

Specifies the point size of the chart label text (see the label property).

This property is in the **Label** property group.

## Legend: Legend group

Properties in this group control the visibility, appearance, and content of the legend.

#### Legend group properties

The group contains the following properties:


- "legendBgColor" on page 336
- "legendBgGradientFlag" on page 336
- "legendVisFlag" on page 336
- "legendTextColor" on page 336
- "legendTextFont" on page 336
- "legendTextHeight" on page 336
- "legendValueMinSpace" on page 336
- "legendValueVisFlag" on page 337
- "legendVisFlag" on page 337

#### legendBgColor

Sets the fill color of the legend inner rectangle. The inner rectangle is smaller than and in front of the legend's background rectangle (see "bgColor" on page 330). The chart label (see "label" on page 334) lies outside of the inner rectangle; the rest of the legend text lies within the inner rectangle.

To set the color, select the **...** button and choose a color from the palette to set the background color of the legend. Close the Color Chooser window when you are done.

This property is in the **Legend** property group.

#### legendBgGradientFlag

Select to display a gradient in the legend inner rectangle (see legendBgColor).

This property is in the **Legend** property group.

#### legendVisFlag

Select to display the legend.

This property is in the **Legend** property group.

#### legendTextColor

Sets the color of the legend text (other than the chart label; see "label" on page 334 and "labelTextColor" on page 335). To set the color, select the **...** button and choose a color from the palette. Close the Color Chooser window when you are done.

This property is in the **Legend** property group.

#### legendTextFont

Sets the font of the legend text (other than the chart label: see "label" on page 334 and "labelTextFont" on page 335). Select an item from the drop down menu.

This property is in the **Legend** property group.

#### legendTextHeight

Specifies the point size of the legend text (other than the chart label; see "label" on page 334 and "labelTextHeight" on page 335)

This property is in the **Legend** property group.

#### legendValueMinSpace

Specifies the minimum number of pixels between values and labels in the legend. This property applies only if legendValueVisFlag is enabled.

This property is in the **Legend** property group.

#### legendValueVisFlag

Select to display the numerical values of your data in the legend.

This property is in the **Legend** property group.

#### legendVisFlag

Select to display the legend.

This property is in the **Legend** property group.

## Legend: Object group

Properties in this group control the visibility and transparency of the legend as a whole. They also control (or reflect) the overall position and dimensions of the legend object. In addition, a property in this group reflects the generated name of this individual legend.

#### Object group properties


- "anchor" on page 337
- "dock" on page 337
- "objHeight" on page 337
- "objName" on page 338
- "objWidth" on page 338
- "objX" on page 338
- "objY" on page 338
- "transparencyPercent" on page 338
- "visFlag" on page 338


#### anchor

Select zero or more of **Top**, **Left**, **Bottom**, and **Right** in order to control the object's placement. The `anchor` property is only applied when the display is resized either by changing the **Background Properties** on the display or by resizing the window in **Layout** mode. If an object has the `dock` property set, the `anchor` property is ignored. See "About resize modes" on page 39.

#### dock

Select **None** (default), **Top**, **Left**, **Bottom**, **Right**, or **Fill** in order to control the object's placement in **Layout** resize mode. See "About resize modes" on page 39.

#### objHeight

Set the height of a chart by entering a value for this property or by dragging a handle of the bounding box that appears when the chart is selected. When you drag a handle of

the bounding box, the displayed value for this property changes to reflect the real-time height of the chart.

This property is in the **Object** property group.

#### objName

An identifier that is generated by the Dashboard Builder. This name can be used by other objects' properties in order to refer to the named chart.

This property is in the **Object** property group.

#### objWidth

Set the width of a chart by entering a value for this property or by dragging a handle of the bounding box that appears when the chart is selected. When you drag a handle of the bounding box, the displayed value for this property changes to reflect the real-time width of the chart.

This property is in the **Object** property group.

#### objX

Sets the X coordinate of the center of this visualization object, relative to the lower left corner of the current dashboard. This value is set automatically when you position the object with the mouse.

This property is in the **Object** property group.

#### objY

Sets the Y coordinate of the center of this visualization object, relative to the lower left corner of the current dashboard. This value is set automatically when you position the object with the mouse.

This property is in the **Object** property group.

#### transparencyPercent

Sets the transparency of this chart.

This property is in the **Object** property group.

#### visFlag

Deselect to make this visualization object invisible in the current dashboard.

This property is in the **Object** property group.

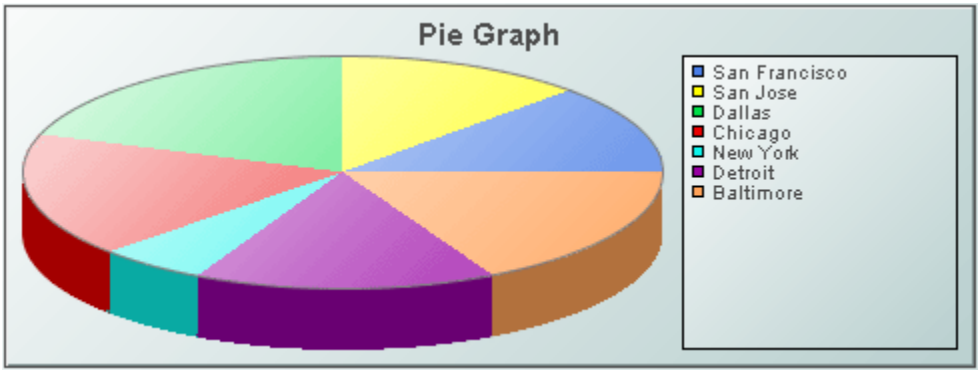
## Pie graph

Pie graphs visualize one or more numerical columns from tabular data. A typical attachment has either one row and multiple numeric columns, or multiple rows with

one numeric column and one non-numeric column (whose values are used as graph labels that uniquely identify each row).

A pie graph can visualize data in either of two ways:

- Column series: The first numeric column of the visualized data is used to populate the wedges in the pie. Each wedge corresponds to a row and displays that row's relative value.
- Row series: The first row of the visualized data is used to populate the wedges in the pie. Each wedge corresponds to a numerical column and displays that column's relative value.




Use the valueTable property to attach data to a pie graph. Use the rowSeriesFlag property to specify row series or column series visualization.

When a pie graph is selected in the Builder canvas, the **Object Class Name** that appears at the top of the **Object Properties** pane is **obj_pie**.

The **Object Properties** panel organizes pie graph properties into the groups below.

## Pie graph: Background group

Properties in this group control the visibility and appearance of the portion of the graph that serves as the background of both the pie and the legend.

### Background group properties

The group contains the following properties:

- "bgBorderColor" on page 340
- "bgBorderFlag" on page 340
- "bgColor" on page 340
- "bgEdgeWidth" on page 340
- "bgGradientColor2" on page 340
- "bgGradientMode" on page 340
- "bgRaisedFlag" on page 341

- "bgRoundness" on page 341
- "bgShadowFlag" on page 341
- "bgStyleFlag" on page 341
- "bgVisFlag" on page 341
- "borderPixels" on page 342


#### bgBorderColor

Sets the color of the border (see `bgBorderFlag`) of the background rectangle. Select the **...** button and choose a color from the palette. Close the Color Chooser window when you are done.

#### bgBorderFlag

Select to display a border around the background rectangle.

This property is in the **Background** property group.

#### bgColor

Sets the background color. Select the ... button and choose a color from the palette. Close the Color Chooser window when you are done.

This property is in the **Background** property group.

#### bgEdgeWidth

Sets the width in pixels of the 3D edge on the background rectangle. This property is only used if `bgBorderFlag` is selected.

This property is in the **Background** property group.

#### bgGradientColor2

Sets the color for the second color in the gradient. The default is white. The `bgColor` property sets the first color in the gradient.

This property is in the **Background** property group.

#### bgGradientMode

Display a gradient in the background rectangle. Select from the following options:

- **None**: No gradient.
- **Diagonal Edge**: Gradient is drawn at a 45 degree angle from the top left to the bottom right corner of the object.
- **Diagonal Center**: Gradient is drawn at a 45 degree angle from the center to the top left and the bottom right corners of the object.

- **Horizontal Edge**: Gradient is drawn horizontally from the top to the bottom of the object.
- **Horizontal Center**: Gradient is drawn horizontally from the center to the top and bottom of the object.
- **Vertical Edge**: Gradient is drawn vertically from the left to the right of the object.
- **Vertical Center**: Gradient is drawn vertically from the center to the left and right of the object.

This property is in the **Background** property group.

#### bgRaisedFlag

Reverses the direction of the gradient, as well as that of the 3D edge if the `bgStyle` selected is 3D Rectangle.

This property is in the **Background** property group.

#### bgRoundness

Sets the arc length of the rounded corners. This property is only available if the `bgStyle` selected is **Round Rectangle**.

The value of `bgRoundness` cannot exceed half the value of the `objWidth` or the `objHeight`. If `bgRoundness` does exceed that value, half of `objWidth` or `objHeight` (whichever is smaller) will be used instead. For example if `objWidth` is `100` and `objHeight` is `50`, then the value of `bgRoundness` cannot exceed `25`. If it does, then half the value of `objHeight` (`25`) will be used instead. This property is in the **Background** property group.

#### bgShadowFlag

Select to display a drop shadow on the background rectangle.

This property is in the **Background** property group.

#### bgStyleFlag

Choose one of the following three options from the drop down menu:

- **Rectangle**: Select to display a background rectangle.
- **3D Rectangle**: Select to display a 3D edge on the background rectangle. If selected, use `bgEdgeWidth` to set the width of the 3D edge.
- **Round Rectangle**: Select to display a background rectangle with rounded edges. If selected, use `bgRoundness` to set the arc length of the rounded corners.

This property is in the **Background** property group.

#### bgVisFlag

Select to display the background rectangle.

This property is in the **Background** property group.

#### borderPixels

Sets the width in pixels of the border between the chart and the edge of the background rectangle.

This property is in the **Background** property group.

## Pie graph: Column group

This group contains one property, columnsToHide, which controls which data-attachment columns are excluded from being used for charted data or labels.

#### columnsToHide

Specify columns from the data attachment to exclude from being used for charted data or labels.

Data from the labelColumnName column will be used for labels even if that column name is also specified in the `columnsToHide` property.

Columns specified in the `columnsToHide` property can still be used in the drillDownColumnSubs property.

This property is in the **Column** property group.

## Pie graph: Data group

Properties in this group control what data appears in the graph, and whether the data appears in column series or row series form.

#### Data group properties

The group contains the following properties:

- "rowSeriesFlag" on page 280
- "valueTable" on page 281

#### rowSeriesFlag

This property controls whether row or column data populates the graph:

- When the `rowSeriesFlag` checkbox is not selected, the first numeric column from your data attachment is used to populate the wedges in the pie. Each wedge corresponds to a row in that column and displays that row's relative value.
- If the `rowSeriesFlag` checkbox is selected, the first row from your data attachment is used to populate the wedges in the pie. Each wedge corresponds to a numerical column in that row and displays that column's relative value. Column names are used in the legend.

This property is in the **Data** property group.

#### valueTable

Attach your data to the `valueTable` property. Right-click on the property name in the Object Properties panel, and select a menu item under **Attach to Data**. A typical attachment has either multiple rows, one numeric column, and one non-numeric column, or one row and multiple numeric columns.

The rowSeriesFlag property controls how row and column data populates the graph:

- When the rowSeriesFlag checkbox is not selected, the first numeric column from your data attachment is used to populate the wedges in the pie. Each wedge corresponds to a row in that column and displays that row's relative value.
- If the rowSeriesFlag checkbox is selected, the first row from your data attachment is used to populate the wedges in the pie. Each wedge corresponds to a numerical column in that row and displays that column's relative value. Column names are used in the legend.

This property is in the **Data** property group.

## Pie graph: Data Format group

Properties on this group control the format of displayed wedge values as well as numerical and date labels.

#### Data Format group properties

The group includes the following properties:


- "labelColumnFormat" on page 343
- "valueFormat" on page 343


#### labelColumnFormat

Sets the format of numeric or date labels displayed in the legend, and in tooltips.

Select or enter the format specification. Use syntax from the Java `DecimalFormat` class for numeric labels, and syntax from the Java `SimpleDateFormat` class for date labels.

To enable tooltips, select the mouseOverFlag.

This property is in the **Data Format** property group.

#### valueFormat

Select or enter the numeric format of wedge values displayed on wedges, in the legend and in tooltips. Use syntax from the Java `DecimalFormat` class. To enable tooltips, select the mouseOverFlag.

This property is in the **Data Format** property group.

## Pie graph: Data Label group

Properties in this group control the labels that are in the legend and in tooltips.

#### Data Label properties

The group contains the following properties:


- "columnDisplayNames" on page 344
- "labelColumnName" on page 344
- "rowLabelVisFlag" on page 344
- "rowNameVisFlag" on page 344


#### columnDisplayNames

Set alternate display names for the columns of the data attached to valueTable. Column names are displayed in the legend, if rowSeriesFlag is selected.

This property is in the **Data Label** property group.

#### labelColumnName

Sets the label column. By default, the label column is the first non-numeric text column in your data attachment, if there is one. Note that the column **apama.instanceID** (contained in all DataView and scenario instance tables) is a non-numerical text column.

Data from the label column is used to label the legend, if rowSeriesFlag is disabled. Data from the label column is used in tooltips, if rowSeriesFlag and mouseOverFlag are enabled.

This property is in the **Data Label** property group.

#### rowLabelVisFlag

Determines whether or not data from the label column is used in the chart legend, when rowSeriesFlag is disabled. See labelColumnName. If `rowLabelVisFlag` is disabled, integer row identifiers are used in the legend.

This property is in the **Data Label** property group.

#### rowNameVisFlag

If your data attachment has no label column (see labelColumnName), select this property to use generated row names in the legend when the rowSeriesFlag is not selected.

This property is in the **Data Label** property group.

## Pie graph: Historian group

Do not use the properties in this group.

#### historyTableName

Do not use this property.

This property is in the **Historian** property group.

#### historyTableRowNameFlag

Do not use this property.

This property is in the **Historian** property group.

## Pie graph: Interaction group

Properties in this group control various forms of interaction between the end user and the graph, including configuring command, drill down, and tooltip interactions.

#### Interaction group properties

The group includes the following properties:


- "command" on page 345
- "commandCloseWindowOnSuccess" on page 346
- "commandConfirm" on page 346
- "confirmText" on page 346
- "drillDownColumnSubs" on page 347
- "drillDownSelectMode" on page 347
- "drillDownTarget" on page 347
- "mouseOverFlag" on page 347


#### command

Assign a command or group of commands to this stock chart by right-clicking on the `command` property name in the **Object Properties** window. Select **Define Command** and choose **SYSTEM**, **APAMA**, or **MULTIPLE**. See "Using the Define Apama Command dialog" on page 226.

Once a command or command group has been assigned to this object, you can activate it from a deployed dashboard or from the Dashboard Builder:

- Dashboard Builder: Double click on the object.
- Web-based deployment: Single click on the object or else right click on it and select **Execute Command** from the popup menu.
- Local deployment: By default, single-click on the object or else right-click on it and select **Execute Command** from the popup menu. To override the default, select **Tools | Options** in the Builder (do this before you generate the deployment package), and

uncheck **Single-Click for Drill Down and Commands** in the **General** tab. This allows the end user to use either a double click or a right click.

When you activate a command, any defined drill down substitutions are performed, and then the command is executed.

If you assign multiple commands, the commands are launched in an arbitrary order, and are executed asynchronously; there is no guarantee that one command will finish before the next one in the sequence starts.

This property is in the **Interaction** property group.

#### commandCloseWindowOnSuccess

Select this property to automatically close the window that initiates a **SYSTEM** command when the command is executed successfully. This applies to **SYSTEM** commands only, and is not supported at all for thin-client, Web-page deployments.

With **APAMA** commands, the window is closed whether or not the command is executed successfully. For **MULTIPLE** commands, the window closes when the first command in the command group succeeds.

This property is in the **Interaction** property group.

#### commandConfirm

By default, when the end user executes a command (see the `command` property), the command confirmation dialog is disabled. To control this option for each individual object, use the `commandConfirm` check box. If confirmation is required for a **MULTIPLE** command group, a single confirmation dialog is presented; if you confirm the execution, all individual commands in the group are executed with no further confirmation. If the you cancel the execution, none of the commands in the group is executed.

You can also override the confirmation status of individual objects with an application-wide policy. Select **Tools | Options** and choose from three confirmation values:

- **Do not confirm**: Indicates that no commands require confirmation (regardless of each object's confirmation status).
- **Confirm all**: Indicates that all commands require confirmation (regardless of each object's confirmation status).
- **Use object confirm flag** (default): Indicates that the confirmation status of each object will determine whether confirmation is required.

This property is in the **Interaction** property group.

#### confirmText

Use this property to write your own text for the confirmation dialog. Otherwise, default text is used. See `commandConfirm`.

This property is in the **Interaction** property group.

#### drillDownColumnSubs

Use this property to direct a dashboard to assign data-table column values to specified dashboard variables when the end user activates a drilldown on this object. In the **Object Properties** window, double-click on `drillDownColumnSubs` in the **Property Name** field to bring up the **Drill Down Column Substitutions** dialog.

The dialog has the following fields and buttons:

- **Substitution String**: Enter the dashboard variable next to the name of the data table column whose value you want assigned to the variable. Press **Enter**.
- **Add Column**: Enter the name of a column and click the **Add Column** button to insert a column into the table.
- **Clear**: Click the **Clear** button to remove all variables listed.

The **Column Name** list is populated based on the table's data attachment. If you have not yet attached the table to data, this list is empty.

Once you have selected which column values to pass in as substitutions, double-click on any element in your object to open a drill down window that displays corresponding values.

This property is in the **Interaction** property group.

#### drillDownSelectMode

Use this property to control how a drill down display is activated. Select one of the following:

- **Anywhere** to activate a drill down display by double-clicking anywhere on the chart.
- **Element Only** to enable a drill down display only when you double-click on an element of the chart, such as a bar or candlestick.

This property is in the **Interaction** property group.

#### drillDownTarget

To specify a drill down display, double click on `drillDownTarget` in the **Property Name** field to bring up the **Drill Down Properties** dialog. See "Drill-Down Specification" on page 535.

This property is in the **Interaction** property group.

#### mouseOverFlag

Select this property to enable tooltips for your pie graph. To display a tooltip, point to a pie wedge with your mouse. The tooltip will contain information from your data attachment about that pie wedge.

This property is in the **Interaction** property group.

## Pie graph: Label group

Properties in this group control the graph's main label (which defaults to **Pie Graph**), including text, alignment, color, font, and size.

### Label group properties

The group includes the following properties:


- "label" on page 348
- "labelTextAlignX" on page 348
- "labelTextColor" on page 348
- "labelTextFont" on page 348
- "labelTextHeight" on page 348


#### label

Specifies the text for the chart label. Click the ellipsis for multi-line text.

The default is **Pie Graph**.

This property is in the **Label** property group.

#### labelTextAlignX

Sets the alignment of the chart label (see the label property). Select **Left**, **Center**, or **Right** from the drop down list.

This property is in the **Label** property group.

#### labelTextColor

Specifies the color of the chart label text (see the label property). Select the **...** button and choose a color from the palette. Close the Color Chooser window when you are done.

This property is in the **Label** property group.

#### labelTextFont

Specifies the font of the chart label text (see the label property). Select an item from drop down list.

This property is in the **Label** property group.

#### labelTextHeight

Specifies the point size of the chart label text (see the label property).

This property is in the **Label** property group.

## Pie graph: Legend group

Properties in this group control the visibility, appearance, and content of the graph legend.

### Legend group properties

The group contains the following properties:


- "legendBgColor" on page 349
- "legendBgGradientFlag" on page 349
- "legendValueVisFlag" on page 349
- "legendVisFlag" on page 349
- "legendWidthPercent" on page 349


### legendBgColor

Select the **...** button and choose a color from the palette to set the background color of the legend. Close the Color Chooser window when you are done.

This property is in the **Legend** property group.

### legendBgGradientFlag

Select to display a gradient in the legend background.

This property is in the **Legend** property group.

### legendValueVisFlag

Select to display the numerical values of your data in the legend.

This property is in the **Legend** property group.

### legendVisFlag

Select to display the legend.

This property is in the **Legend** property group.

### legendWidthPercent

Set the percent of the total width of the object used for the legend.

This property is in the **Legend** property group.

## Pie graph: Object group

Properties in this group control the visibility and transparency of the graph as a whole. They also control (or reflect) the overall position and dimensions of the graph. In addition, a property in this group reflects the generated name of this individual graph.

#### Object group properties

This group contains the following properties:


- "anchor" on page 350
- "dock" on page 350
- "objHeight" on page 350
- "objName" on page 350
- "objWidth" on page 350
- "objX" on page 351
- "objY" on page 351
- "transparencyPercent" on page 351
- "visFlag" on page 351


#### anchor

Select zero or more of **Top**, **Left**, **Bottom**, and **Right** in order to control the object's placement. The `anchor` property is only applied when the display is resized either by changing the **Background Properties** on the display or by resizing the window in **Layout** mode. If an object has the `dock` property set, the `anchor` property is ignored. See "About resize modes" on page 39.

#### dock

Select **None** (default), **Top**, **Left**, **Bottom**, **Right**, or **Fill** in order to control the object's placement in **Layout** resize mode. See "About resize modes" on page 39.

#### objHeight

Set the height of a chart by entering a value for this property or by dragging a handle of the bounding box that appears when the chart is selected. When you drag a handle of the bounding box, the displayed value for this property changes to reflect the real-time height of the chart.

This property is in the **Object** property group.

#### objName

An identifier that is generated by the Dashboard Builder. This name can be used by other objects' properties in order to refer to the named chart.

This property is in the **Object** property group.

#### objWidth

Set the width of a chart by entering a value for this property or by dragging a handle of the bounding box that appears when the chart is selected. When you drag a handle of

the bounding box, the displayed value for this property changes to reflect the real-time width of the chart.

This property is in the **Object** property group.

#### objX

Sets the X coordinate of the center of this visualization object, relative to the lower left corner of the current dashboard. This value is set automatically when you position the object with the mouse.

This property is in the **Object** property group.

#### objY

Sets the Y coordinate of the center of this visualization object, relative to the lower left corner of the current dashboard. This value is set automatically when you position the object with the mouse.

This property is in the **Object** property group.

#### transparencyPercent

Sets the transparency of this chart.

This property is in the **Object** property group.

#### visFlag

Deselect to make this visualization object invisible in the current dashboard.

This property is in the **Object** property group.

## Pie graph: Wedge group

Properties in this group control the appearance of the pie's wedges, including thickness, gradient effect, and color.

#### Wedge group properties

This group contains the following properties:

- "pieThickness" on page 351
- "wedgeGradientFlag" on page 352
- "wedgeProperties" on page 352

#### pieThickness

Sets the thickness of the wedges in pixels.

This property is in the **Wedge** property group.

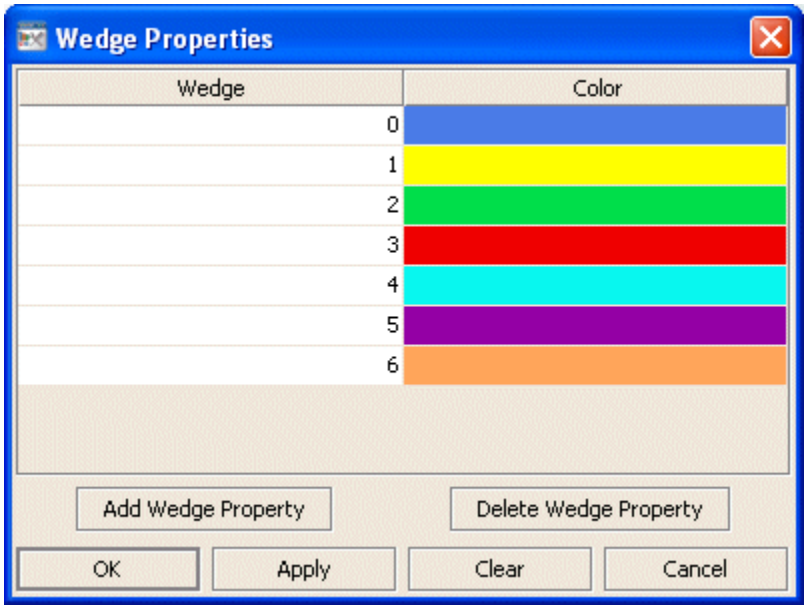
#### wedgeGradientFlag

Select the box to enable the gradient effect in the wedges.

This property is in the **Wedge** property group.

#### wedgeProperties

Use this property to assign a color to each wedge in a pie graph. In the Object Properties window, double-click on `wedgeProperties` in the **Property Name** field to bring up the Wedge Properties dialog.




**Note:** Before you assign attributes to wedges in your pie graph, it is recommended that you first attach the pie graph to data.

The Wedge Properties dialog has two columns of fields:

- **Wedge**: Each wedge from the pie graph is listed.
- **Color**: Select the ellipsis button in the and choose a color from the palette. Close the Color Chooser window.

The dialog has the following buttons:

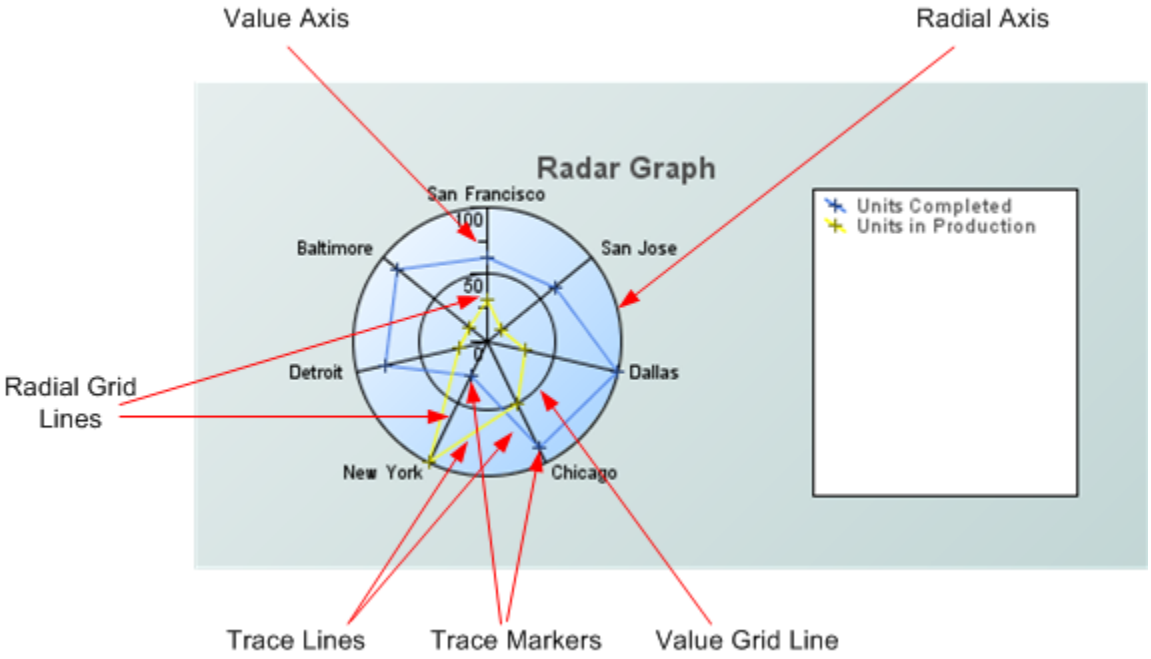
- **Add Wedge Property**: Click to add a wedge entry field. The data for the wedge does not have to be available yet. You may consider adding and assigning attributes to more wedges than your data currently needs for when you have more data to show.
- **Delete Wedge Property**: Removes the last wedge entry field from the Wedge Properties dialog.
- **OK**: Applies values and closes the dialog.
- **Apply**: Applies values without closing the dialog.

- **Reset**: Resets all fields to last values applied.
- **Clear**: Clears all fields. Detaches object from data source (once Apply or OK is selected).
- **Cancel**: Closes the dialog with last values applied.

This property is in the **Wedge** property group.

## Radar graph

Radar graphs visualize tabular data that has one or more numerical columns. Typically, the visualized data also has one non-numerical column whose values are used as graph labels that uniquely identify each row.




A radar graph can visualize data in either of two ways:

- Row series visualization: The graph displays one radial grid line for each numeric column of the visualized data, and one trace for each row of the data. A given trace intersects a given radial grid line at a distance (from the graph's center) that is proportional to the value of the grid line's corresponding column for the trace's corresponding row. A marker is displayed at the point of intersection.
- Column series visualization: The graph displays one radial grid line for each row of your data attachment, and one trace for each numeric column of your data attachment. A given trace intersects a given radial grid line at a distance (from the graph's center) that is proportional to the value of the trace's corresponding column for the grid line's corresponding row. A marker is displayed at the point of intersection.

Use the valueTable property to attach data to a radar graph. Use the rowSeriesFlag property to specify row series or column series visualization.

When a radar graph is selected in the Builder canvas, the **Object Class Name** that appears at the top of the **Object Properties** pane is **obj_radar**.

The **Object Properties** panel organizes radar graph properties into the groups below.

## Radar graph: Alert group

Properties in this group allow you to specify changes in the appearance of trace lines and markers that signal changes in the status of specified data elements. You can specify threshold values (see valueHighAlarm, valueHighWarning, valueLowAlarm, and valueHighWarning) or attach a data table to valueAlarmStatusTable that indicates the status of each element of the table that is attached to valueTable.

### Alert group properties

This group includes the following properties:


- "valueAlarmStatusTable" on page 355
- "valueHighAlarm" on page 355
- "valueHighAlarmEnabledFlag" on page 355
- "valueHighAlarmLineVisFlag" on page 356
- "valueHighAlarmMarkColor" on page 356
- "valueHighAlarmMarkStyle" on page 356
- "valueHighWarning" on page 356
- "valueHighWarningEnabledFlag" on page 356
- "valueHighWarningLineVisFlag" on page 357
- "valueHighWarningMarkColor" on page 357
- "valueHighWarningMarkStyle" on page 357
- "valueLowAlarm" on page 357
- "valueLowAlarmEnabledFlag" on page 357
- "valueLowAlarmLineVisFlag" on page 358
- "valueLowAlarmMarkColor" on page 358
- "valueLowAlarmMarkStyle" on page 358
- "valueLowWarning" on page 358
- "valueLowWarningEnabledFlag" on page 358
- "valueLowWarningLineVisFlag" on page 359

- "valueLowWarningMarkColor" on page 359
- "valueLowWarningMarkStyle" on page 359

#### valueAlarmStatusTable

Attach an alarm table containing status indexes to this property in order to enable rule based alarm statuses for trace markers. The table attached to `valueAlarmStatusTable` must have the same number of rows and columns as valueTable. For each data element in valueTable, the status index at the corresponding position in `valueAlarmStatusTable` is used to set the alarm status of the marker that represents the data element.

Following are the valid indexes are:

- **0**: Use normal marker color and style. See "traceProperties" on page 297.
- **1**: Use low alarm marker color and style. See "valueLowAlarmMarkColor" on page 358 and "valueLowAlarmMarkStyle" on page 358.
- **2**: Use low warning marker color and style. See "valueLowWarningMarkColor" on page 359 and "valueLowWarningMarkStyle" on page 359.
- **3**: Use high warning marker color and style. See "valueHighWarningMarkColor" on page 357 and "valueHighWarningMarkStyle" on page 357.
- **4**: Use high alarm marker color and style. See "valueHighAlarmMarkColor" on page 356 and "valueHighAlarmMarkStyle" on page 356.
- **-1**: Determine marker color and style by comparing the value to the enabled alarm thresholds. See "valueHighAlarm" on page 355, "valueHighWarning" on page 356, "valueLowAlarm" on page 357, and "valueLowWarning" on page 358.

If no data is attached to `valueAlarmStatusTable`, the alarm status for a trace marker is determined by comparing the marker's value to the enabled thresholds. See valueHighAlarm, valueHighWarning, valueLowAlarm, and valueLowWarning.

This property is in the **Alert** property group.

#### valueHighAlarm

Specifies the threshold value used by valueHighAlarmLineVisFlag, valueHighAlarmMarkColor, and valueHighAlarmMarkStyle.

This property is in the **Alert** property group.

#### valueHighAlarmEnabledFlag

Select to enable the high alarm threshold. See valueHighAlarm.

This property is in the **Alert** property group.

#### valueHighAlarmLineVisFlag

Select to display a dashed line at the high alarm threshold. The color of the line is set to valueHighAlarmMarkColor. This line is displayed only if valueHighAlarmEnabledFlag is selected.

This property is in the **Alert** property group.

#### valueHighAlarmMarkColor

When a trace marker's value is greater than or equal to valueHighAlarm, the marker changes to `valueHighAlarmMarkColor` and valueHighWarningMarkStyle, provided valueHighAlarmEnabledFlag is selected and no data is attached to valueAlarmStatusTable.

If data is attached to valueAlarmStatusTable, a marker changes to `valueHighAlarmMarkColor` and valueHighAlarmMarkStyle when the marker's corresponding element in the attached alarm status table is 4.

If data is attached to valueAlarmStatusTable, and the marker's corresponding element in the attached alarm status table is -1, marker color and style behave as if no data were attached to valueAlarmStatusTable.

This property is in the **Alert** property group.

#### valueHighAlarmMarkStyle

When a trace marker's value is greater than or equal to `valueHighAlarm`, the marker changes to valueHighAlarmMarkColor and `valueHighAlarmMarkStyle`, provided valueHighAlarmEnabledFlag is selected and no data is attached to valueAlarmStatusTable.

If data is attached to valueAlarmStatusTable, a marker changes to valueHighAlarmMarkColor and `valueHighAlarmMarkStyle` when the marker's corresponding element in the attached alarm status table is 4.

If data is attached to valueAlarmStatusTable, and the marker's corresponding element in the attached alarm status table is -1, marker color and style behave as if no data were attached to valueAlarmStatusTable.

This property is in the **Alert** property group.

#### valueHighWarning

Specifies the threshold value used by valueHighWarningLineVisFlag, valueHighWarningMarkColor, and valueHighWarningMarkStyle.

This property is in the **Alert** property group.

#### valueHighWarningEnabledFlag

Select to enable the high warning threshold. See valueHighWarning.

This property is in the **Alert** property group.

#### valueHighWarningLineVisFlag

Select to display a dashed line at the high warning threshold. The color of the line is set to valueHighWarningMarkColor. This line is displayed only if valueHighWarningEnabledFlag is selected.

This property is in the **Alert** property group.

#### valueHighWarningMarkColor

When a trace marker's value is greater than or equal to valueHighWarning but less than valueHighAlarm, the marker changes to `valueHighWarningMarkColor` and valueHighAlarmMarkStyle, provided valueHighAlarmEnabledFlag is selected and no data is attached to valueAlarmStatusTable.

If data is attached to valueAlarmStatusTable, a marker changes to `valueHighWarningMarkColor` and valueHighWarningMarkStyle when the marker's corresponding element in the attached alarm status table is 3.

If data is attached to valueAlarmStatusTable, and the marker's corresponding element in the attached alarm status table is -1, marker color and style behave as of no data were attached to valueAlarmStatusTable.

This property is in the **Alert** property group.

#### valueHighWarningMarkStyle

When a trace marker's value is greater than or equal to valueHighWarning but less than valueHighAlarm, the marker changes to valueHighWarningMarkColor and `valueHighWarningMarkStyle`, provided valueHighWarningEnabledFlag is selected and no data is attached to valueAlarmStatusTable.

If data is attached to valueAlarmStatusTable, a marker changes to valueHighWarningMarkColor and `valueHighWarningMarkStyle` when the marker's corresponding element in the attached alarm status table is 3.

If data is attached to valueAlarmStatusTable, and the marker's corresponding element in the attached alarm status table is -1, marker color and style behave as of no data were attached to valueAlarmStatusTable.

This property is in the **Alert** property group.

#### valueLowAlarm

Specifies the threshold value used by valueLowAlarmLineVisFlag, valueLowAlarmMarkColor, and valueLowWarningMarkStyle.

This property is in the **Alert** property group.

#### valueLowAlarmEnabledFlag

Select to enable the low alarm threshold. See valueLowAlarm.

This property is in the **Alert** property group.

#### valueLowAlarmLineVisFlag

Select to display a dashed line at the low alarm threshold. The color of the line is set to valueLowAlarmMarkColor. This line is displayed only if valueLowAlarmEnabledFlag is selected.

This property is in the **Alert** property group.

#### valueLowAlarmMarkColor

When a trace marker's value is less than or equal to valueLowAlarm, the marker changes to `valueLowAlarmMarkColor` and valueLowAlarmMarkStyle, provided valueLowAlarmEnabledFlag is selected and no data is attached to valueAlarmStatusTable.

If data is attached to valueAlarmStatusTable, a marker changes to `valueLowAlarmMarkColor` and valueLowAlarmMarkStyle when the marker's corresponding element in the attached alarm status table is 1.

If data is attached to valueAlarmStatusTable, and the marker's corresponding element in the attached alarm status table is -1, marker color and style behave as of no data were attached to valueAlarmStatusTable.

This property is in the **Alert** property group.

#### valueLowAlarmMarkStyle

When a trace marker's value is less than or equal to valueLowAlarm, the marker changes to valueLowAlarmMarkColor and `valueLowAlarmMarkStyle`, provided valueLowAlarmEnabledFlag is selected and no data is attached to valueAlarmStatusTable.

If data is attached to valueAlarmStatusTable, a marker changes to valueLowAlarmMarkColor and `valueLowAlarmMarkStyle` when the marker's corresponding element in the attached alarm status table is 1.

If data is attached to valueAlarmStatusTable, and the marker's corresponding element in the attached alarm status table is -1, marker color and style behave as of no data were attached to valueAlarmStatusTable.

This property is in the **Alert** property group.

#### valueLowWarning

Specifies the threshold value used by valueLowWarningLineVisFlag, valueLowWarningMarkColor, and valueLowWarningMarkStyle.

This property is in the **Alert** property group.

#### valueLowWarningEnabledFlag

Select to enable the low warning threshold. See valueLowWarning.

This property is in the **Alert** property group.

#### valueLowWarningLineVisFlag

Select to display a dashed line at the low warning threshold. The color of the line is set to valueLowWarningMarkColor. This line is displayed only if valueLowWarningEnabledFlag is selected.

This property is in the **Alert** property group.

#### valueLowWarningMarkColor

When a trace marker's value is less than or equal to valueLowWarning but greater than valueLowAlarm, the marker changes to `valueLowWarningMarkColor` and valueLowAlarmMarkStyle, provided valueLowWarningEnabledFlag is selected and no data is attached to valueAlarmStatusTable.

If data is attached to valueAlarmStatusTable, a marker changes to `valueLowWarningMarkColor` and valueLowWarningMarkStyle when the marker's corresponding element in the attached alarm status table is 2.

If data is attached to valueAlarmStatusTable, and the marker's corresponding element in the attached alarm status table is -1, marker color and style behave as of no data were attached to valueAlarmStatusTable.

This property is in the **Alert** property group.

#### valueLowWarningMarkStyle

When a trace marker's value is less than or equal to valueLowWarning but greater than valueLowAlarm, the marker changes to valueLowWarningMarkColor and `valueLowWarningMarkStyle`, provided valueLowWarningEnabledFlag is selected and no data is attached to valueAlarmStatusTable.

If data is attached to valueAlarmStatusTable, a marker changes to valueLowWarningMarkColor and `valueLowWarningMarkStyle` when the marker's corresponding element in the attached alarm status table is 2.

If data is attached to valueAlarmStatusTable, and the marker's corresponding element in the attached alarm status table is -1, marker color and style behave as of no data were attached to valueAlarmStatusTable.

This property is in the **Alert** property group.

## Radar graph: Background group

Properties in this group control the visibility and appearance of the portion of the graph that serves as the background of both the plot area and legend.

#### Background group properties

The group contains the following properties:

- "bgBorderColor" on page 360

- "bgBorderFlag" on page 360
- "bgColor" on page 360
- "bgEdgeWidth" on page 360
- "bgGradientColor2" on page 360
- "bgGradientMode" on page 361
- "bgRaisedFlag" on page 361
- "bgRoundness" on page 361
- "bgShadowFlag" on page 361
- "bgStyleFlag" on page 361
- "bgVisFlag" on page 362
- "borderPixels" on page 362


#### bgBorderColor

Sets the color of the border (see `bgBorderFlag`) of the background rectangle. Select the **...** button and choose a color from the palette. Close the Color Chooser window when you are done.

#### bgBorderFlag

Select to display a border around the background rectangle.

This property is in the **Background** property group.

#### bgColor

Sets the background color. Select the **...** button and choose a color from the palette. Close the Color Chooser window when you are done.

This property is in the **Background** property group.

#### bgEdgeWidth

Sets the width in pixels of the 3D edge on the background rectangle. This property is only used if `bgBorderFlag` is selected.

This property is in the **Background** property group.

#### bgGradientColor2

Sets the color for the second color in the gradient. The default is white. The `bgColor` property sets the first color in the gradient.

This property is in the **Background** property group.

#### bgGradientMode

Display a gradient in the background rectangle. Select from the following options:

- **None**: No gradient.
- **Diagonal Edge**: Gradient is drawn at a 45 degree angle from the top left to the bottom right corner of the object.
- **Diagonal Center**: Gradient is drawn at a 45 degree angle from the center to the top left and the bottom right corners of the object.
- **Horizontal Edge**: Gradient is drawn horizontally from the top to the bottom of the object.
- **Horizontal Center**: Gradient is drawn horizontally from the center to the top and bottom of the object.
- **Vertical Edge**: Gradient is drawn vertically from the left to the right of the object.
- **Vertical Center**: Gradient is drawn vertically from the center to the left and right of the object.

This property is in the **Background** property group.

#### bgRaisedFlag

Reverses the direction of the gradient, as well as that of the 3D edge if the `bgStyle` selected is 3D Rectangle.

This property is in the **Background** property group.

#### bgRoundness

Sets the arc length of the rounded corners. This property is only available if the `bgStyle` selected is **Round Rectangle**.

The value of `bgRoundness` cannot exceed half the value of the `objWidth` or the `objHeight`. If `bgRoundness` does exceed that value, half of `objWidth` or `objHeight` (whichever is smaller) will be used instead. For example if `objWidth` is `100` and `objHeight` is `50`, then the value of `bgRoundness` cannot exceed `25`. If it does, then half the value of `objHeight` (`25`) will be used instead. This property is in the **Background** property group.

#### bgShadowFlag

Select to display a drop shadow on the background rectangle.

This property is in the **Background** property group.

#### bgStyleFlag

Choose one of the following three options from the drop down menu:

- **Rectangle**: Select to display a background rectangle.

- **3D Rectangle**: Select to display a 3D edge on the background rectangle. If selected, use `bgEdgeWidth` to set the width of the 3D edge.
- **Round Rectangle**: Select to display a background rectangle with rounded edges. If selected, use `bgRoundness` to set the arc length of the rounded corners.

This property is in the **Background** property group.

#### bgVisFlag

Select to display the background rectangle.

This property is in the **Background** property group.

#### borderPixels

Sets the width in pixels of the border between the chart and the edge of the background rectangle.

This property is in the **Background** property group.

## Radar graph: Column group

This group contains one property, columnsToHide, which controls which data-attachment columns are excluded from being used for plotted data or labels.

#### columnsToHide

Specify columns from the data attachment to exclude from being used for plotted data or labels. Data from the labelColumnName column will be used for labels even if that column name is also specified in the `columnsToHide` property. Columns specified in the `columnsToHide` property can still be used in the drillDownColumnSubs property.

This property is in the **Column** property group.

## Radar graph: Data group

Properties in this group control what data appears in the graph, as well as whether the data appears in column series or row series form.

The group contains the following properties:


- "rowSeriesFlag" on page 362
- "valueDivisor" on page 363
- "valueDivisor" on page 363
- "valueMin" on page 363
- "valueTable" on page 364


#### rowSeriesFlag

The `rowSeriesFlag` property controls how data populates the graph:

- If the `rowSeriesFlag` is enabled, the graph displays one radial grid line for each numeric column of your data attachment (see valueTable), and one trace for each row of your data attachment. A given trace intersects a given radial grid line at a distance (from the graph's center) that is proportional to the value of the grid line's corresponding column for the trace's corresponding row. A marker is displayed at the point of intersection.

If the attachment has a label column (see labelColumnName) and `rowLabelVisFlag` is selected, values from that column are used as legend labels. If `radialAxisLabelVisFlag` is enabled, the numerical column names appear as labels along the radial axis.

- If the `rowSeriesFlag` is disabled, the graph displays one radial grid line for each row of your data attachment (see valueTable), and one trace for each numeric column of your data attachment. A given trace intersects a given radial grid line at a distance (from the graph's center) that is proportional to the value of the trace's corresponding column for the grid line's corresponding row. A marker is displayed at the point of intersection.

If the attachment has a label column (see labelColumnName) and both rowLabelVisFlag and radialAxisLabelVisFlag are enabled, values from that column appear as labels along the radial axis. Numerical column names are used as legend labels.

This property is in the **Data** property group.

#### valueDivisor

Specifies a value by which to divide data table values in order to arrive at the plotted value for this chart.

The default value is 1. If this property is set to 0, the dashboard uses 1 as the divisor.

This property is in the **Data** property group.

#### valueMax

The valueMin and `valueMax` properties control the range of the value axis if valueAxisAutoScaleMode is set to **Off**. In this case, the chart origin (the bottom of the value axis) is labeled with valueMin. The intersection of the value axis and the radial axis (the top of the value axis) is labeled with `valueMax`.

In addition, if valueAxisAutoScaleMode is set to **On - Include Min/Max**, the dashboard calculates the smallest x-axis range that includes both valueMin and `valueMax` as well as all plotted points.

This property is in the **Data** property group.

#### valueMin

The `valueMin` and valueMax properties control the range of the value axis if valueAxisAutoScaleMode is set to **Off**. In this case, the chart origin (the bottom of the value axis) is labeled with `valueMin`. The intersection of the value axis and the radial axis (the top of the value axis) is labeled with valueMax.

In addition, if valueAxisAutoScaleMode is set to **On - Include Min/Max**, the dashboard calculates the smallest x-axis range that includes both `valueMin` and valueMax as well as all plotted points.

This property is in the **Data** property group.

#### valueTable

Attach your data to the `valueTable` property. Right-click on the property name in the Object Properties panel, and select a menu item under **Attach to Data**.

The attached data table should have one or more numerical columns. Typically, the data attachment also has one non-numerical column, the label column (see labelColumnName) whose values uniquely identify each row (that is, no two rows of the table have the same value for the label column).

The rowSeriesFlag property controls how data populates the graph:

- If the rowSeriesFlag is enabled, the graph displays one radial grid line for each numeric column of your data attachment, and one trace for each row of your data attachment. A given trace intersects a given radial grid line at a distance (from the graph's center) that is proportional to the value of the grid line's corresponding column for the trace's corresponding row. A marker is displayed at the point of intersection.

If the attachment has a label column and rowLabelVisFlag is selected, values from that column are used in the legend in order to identify each trace. If radialAxisLabelVisFlag is enabled, the numerical column names appear as labels along the radial axis.

- If the rowSeriesFlag is disabled, the graph displays one radial grid line for each row of your data attachment, and one trace for each numeric column of your data attachment. A given trace intersects a given radial grid line at a distance (from the graph's center) that is proportional to the value of the trace's corresponding column for the grid line's corresponding row. A marker is displayed at the point of intersection.

If the attachment has a label column and both rowLabelVisFlag and radialAxisVisFlag are enabled, values from that column appear as labels along the radial axis. Numerical column names are used in the legend in order to identify each trace.

This property is in the **Data** property group.

## Radar graph: Data Format group

Properties on this group control the format of displayed values as well as numerical and date labels.

The group includes the following properties:


- "labelColumnFormat" on page 365
- "valueFormat" on page 365

#### labelColumnFormat

Sets the format of numeric or date labels displayed in the legend, along the radial axis, and in tooltips.

Select or enter the format specification. Use syntax based on the Java `DecimalFormat` class for numeric labels, and syntax based on the Java `SimpleDateFormat` class for date labels.

To enable tooltips, select the mouseOverFlag.

This property is in the **Data Format** property group.

#### valueFormat

Sets the numeric format of trace values displayed in tooltips.

Select or enter a format. Use syntax from the Java **DecimalFormat** class. To enable tooltips, select the mouseOverFlag property.

This property is in the **Data Format** property group.

## Radar graph: Data Label group

Properties in this group control the labels that are used along the radial axis or in the legend.

#### Data Label group properties

The group contains the following properties:


- "columnDisplayNames" on page 365
- "labelColumnName" on page 365
- "rowLabelVisFlag" on page 366
- "rowNameVisFlag" on page 366


#### columnDisplayNames

Set alternate display names for the columns of the data attached to valueTable. Column names label the radial axes or are used in the legend, depending on whether or not rowSeriesFlag is selected.

This property is in the **Data Label** property group.

#### labelColumnName

Sets the label column. By default, the label column is the first non-numeric text column in your data attachment, if there is one. Data from the label column either appears as labels along the radial axis or else is used in the legend, depending on whether rowSeriesFlag is enabled.

If both rowSeriesFlag and rowLabelVisFlag are enabled, data from the label column is used in the legend.

If rowSeriesFlag is not enabled and both rowLabelVisFlag and radialAxisLabelVisFlag are enabled, data from the label column appears as labels the radial axis.

This property is in the **Data Label** property group.

#### rowLabelVisFlag

Determines whether or not data from the label column is used in chart labels. See labelColumnName. If `rowLabelVisFlag` is disabled, integer row identifiers either appear as labels along the radial axis (if rowSeriesFlag is disabled and radialAxisLabelVisFlag is enabled) or else are used in the legend (if rowSeriesFlag is enabled).

This property is in the **Data Label** property group.

#### rowNameVisFlag

If your data attachment has no label column (see labelColumnName), select this property to use generated row names in chart labels.

This property is in the **Data Label** property group.

## Radar graph: Historian group

Do not use the properties in this group.

#### historyTableName

Do not use this property.

This property is in the **Historian** property group.

#### historyTableRowNameFlag

Do not use this property.

This property is in the **Historian** property group.

## Radar graph: Interaction group

Properties in this group control various forms of interaction between the end user and the graph, including command, drill down, and tooltip interactions.

#### Interaction group properties

The group includes the following properties:

- "command" on page 367
- "commandCloseWindowOnSuccess" on page 367

- "commandConfirm" on page 368
- "confirmText " on page 368
- "drillDownColumnSubs" on page 368
- "drillDownSelectMode" on page 369
- "drillDownTarget" on page 369
- "mouseOverFlag" on page 369


#### command

Assign a command or group of commands to this stock chart by right-clicking on the `command` property name in the Object Properties window. Select **Define Command** and choose **SYSTEM**, **APAMA**, or **MULTIPLE**. See "Using the Define Apama Command dialog" on page 226.

Once a command or command group has been assigned to this object, you can activate it from a deployed dashboard or from the Dashboard Builder:

- Dashboard Builder: Double click on the object.
- Web-based deployment: Single click on the object or else right click on it and select **Execute Command** from the popup menu.
- Local deployment: By default, single-click on the object or else right-click on it and select **Execute Command** from the popup menu. To override the default, select **Tools > Options** in the Builder (do this before you generate the deployment package), and uncheck **Single-Click for Drill Down and Commands** in the **General** tab. This allows the end user to use either a double click or a right click.

When you activate a command, any defined drill down substitutions are performed, and then the command is executed.

If you assign multiple commands, the commands are launched in an arbitrary order, and are executed asynchronously; there is no guarantee that one command will finish before the next one in the sequence starts.

This property is in the **Interaction** property group.

#### commandCloseWindowOnSuccess

Select this property to automatically close the window that initiates a **SYSTEM** command when the command is executed successfully. This applies to **SYSTEM** commands only, and is not supported at all for thin-client, Web-page deployments.

With **APAMA** commands, the window is closed whether or not the command is executed successfully. For **MULTIPLE** commands, the window closes when the first command in the command group succeeds.

This property is in the **Interaction** property group.

#### commandConfirm

By default, when the end user executes a command (see the `command` property), the command confirmation dialog is disabled. To control this option for each individual object, use the `commandConfirm` check box. If confirmation is required for a **MULTIPLE** command group, a single confirmation dialog is presented; if you confirm the execution, all individual commands in the group are executed with no further confirmation. If the you cancel the execution, none of the commands in the group is executed.

You can also override the confirmation status of individual objects with an application-wide policy. Select **Tools > Options** and choose from three confirmation values:

- **Do not confirm**: Indicates that no commands require confirmation (regardless of each object's confirmation status).
- **Confirm all**: Indicates that all commands require confirmation (regardless of each object's confirmation status).
- **Use object confirm flag** (default): Indicates that the confirmation status of each object will determine whether confirmation is required.

This property is in the **Interaction** property group.

#### confirmText

Use this property to write your own text for the confirmation dialog. Otherwise, default text is used. See `commandConfirm`.

This property is in the **Interaction** property group.

#### drillDownColumnSubs

Use this property to direct a dashboard to assign data-table column values to specified dashboard variables when the end user activates a drilldown on this object. In the Object Properties window, double-click on `drillDownColumnSubs` in the **Property Name** field to bring up the Drill Down Column Substitutions dialog.

The dialog has the following fields and buttons:

- **Substitution String**: Enter the dashboard variable next to the name of the data table column whose value you want assigned to the variable. Press Enter.
- **Add Column**: Enter the name of a column and click the **Add Column** button to insert a column into the table.
- **Clear**: Click the **Clear** button to remove all variables listed.

The **Column Name** list is populated based on the table's data attachment. If you have not yet attached the table to data, this list is empty.

Once you have selected which column values to pass in as substitutions, double-click on any element in your object to open a drill down window that displays corresponding values.

This property is in the **Interaction** property group.

#### drillDownSelectMode

Use this property to control how a drill down display is activated. Select one of the following:

- **Anywhere** to activate a drill down display by double-clicking anywhere on the chart.
- **Element Only** to enable a drill down display only when you double-click on an element of the chart, such as a bar or candlestick.

This property is in the **Interaction** property group.

#### drillDownTarget

To specify a drill down display, double click on `drillDownTarget` in the **Property Name** field to bring up the Drill Down Properties dialog. See "Drill-Down Specification" on page 535.

This property is in the **Interaction** property group.

#### mouseOverFlag

Select this property to enable tooltips for your radar graph. To display a tooltip, point to a trace marker with your mouse. The tooltip will contain information from your data attachment about that marker.

This property is in the **Interaction** property group.

## Radar graph: Label group

Properties in this group control the graph's main label (which defaults to **Radar Graph**), including text, alignment, color, font, and size.

#### Label group properties

The group includes the following properties:

- "label" on page 369
- "labelTextAlignX" on page 370
- "labelTextColor" on page 370
- "labelTextFont" on page 370
- "labelTextHeight" on page 370

#### label

Specifies the text for the chart label. Click the ellipsis for multi-line text.

The default is **Radar Graph**.

This property is in the **Label** property group.

#### labelTextAlignX

Sets the alignment of the chart label (see the label property). Select **Left**, **Center**, or **Right** from the drop down list.

This property is in the **Label** property group.

#### labelTextColor

Specifies the color of the chart label text (see the label property). Select the ... button and choose a color from the palette. Close the Color Chooser window when you are done.

This property is in the **Label** property group.

#### labelTextFont

Specifies the font of the chart label text (see the label property). Select an item from drop down list.

This property is in the **Label** property group.

#### labelTextHeight

Specifies the point size of the chart label text (see the label property).

This property is in the **Label** property group.

## Radar graph: Legend group

Properties in this group control the visibility and appearance of the graph legend.

#### Legend group properties

The group contains the following properties:

- "legendBgColor" on page 370
- "legendBgGradientFlag" on page 370
- "legendVisFlag" on page 371
- "legendWidthPercent" on page 371

#### legendBgColor

Select the **...** button and choose a color from the palette to set the background color of the legend. Close the Color Chooser window when you are done.

This property is in the **Legend** property group.

#### legendBgGradientFlag

Select to display a gradient in the legend background.

This property is in the **Legend** property group.

#### legendVisFlag

Select to display the legend.

This property is in the **Legend** property group.

#### legendWidthPercent

Set the percent of the total width of the object used for the legend.

This property is in the **Legend** property group.

## Radar graph: Marker group

Properties in this group control the appearance of trace markers (but see also "Radar graph: Trace group" on page 375).

#### Marker group properties

The group contains the following properties:

- "markDefaultSize" on page 371
- "markScaleMode" on page 371

#### markDefaultSize

Sets the size of the trace markers in pixels. Supply an integer value that is between 1 and 18, inclusive.

This property is in the **Marker** property group.

#### markScaleMode

Sets the scale mode for trace marks. Select one of the following from the drop down menu:

- **No Scale**: All marks, across and within traces, are the same size.
- **Scale by Trace**: Scale marks according to the trace in which they reside, that is, marks in the first trace are the largest, across all traces, and the marks in the last trace are the smallest.
- **Scale Within Trace**: Scale marks according to the relative order of the data within each trace.

This property is in the **Marker** property group.

## Radar graph: Object group

Properties in this group control the visibility and transparency of the graph as a whole. They also control (or reflect) the overall position and dimensions of the graph. In addition, a property in this group reflects the generated name of this individual graph.

#### Object group properties

This group contains the following properties:


- "anchor" on page 372
- "dock" on page 372
- "objHeight" on page 372
- "objName" on page 372
- "objWidth" on page 372
- "objX" on page 373
- "objY" on page 373
- "transparencyPercent" on page 373
- "visFlag" on page 373


#### anchor

Select zero or more of **Top**, **Left**, **Bottom**, and **Right** in order to control the object's placement. The `anchor` property is only applied when the display is resized either by changing the **Background Properties** on the display or by resizing the window in **Layout** mode. If an object has the `dock` property set, the `anchor` property is ignored. See "About resize modes" on page 39.

#### dock

Select **None** (default), **Top**, **Left**, **Bottom**, **Right**, or **Fill** in order to control the object's placement in **Layout** resize mode. See "About resize modes" on page 39.

#### objHeight

Set the height of a chart by entering a value for this property or by dragging a handle of the bounding box that appears when the chart is selected. When you drag a handle of the bounding box, the displayed value for this property changes to reflect the real-time height of the chart.

This property is in the **Object** property group.

#### objName

An identifier that is generated by the Dashboard Builder. This name can be used by other objects' properties in order to refer to the named chart.

This property is in the **Object** property group.

#### objWidth

Set the width of a chart by entering a value for this property or by dragging a handle of the bounding box that appears when the chart is selected. When you drag a handle of

the bounding box, the displayed value for this property changes to reflect the real-time width of the chart.

This property is in the **Object** property group.

#### objX

Sets the X coordinate of the center of this visualization object, relative to the lower left corner of the current dashboard. This value is set automatically when you position the object with the mouse.

This property is in the **Object** property group.

#### objY

Sets the Y coordinate of the center of this visualization object, relative to the lower left corner of the current dashboard. This value is set automatically when you position the object with the mouse.

This property is in the **Object** property group.

#### transparencyPercent

Sets the transparency of this chart.

This property is in the **Object** property group.

#### visFlag

Deselect to make this visualization object invisible in the current dashboard.

This property is in the **Object** property group.

## Radar graph: Plot Area group

Properties in this group control the appearance of the plot area, the rectangular area that serves as background for the axes, grid lines, and trace lines (but not for the legend or radial axis labels; see "Radar graph: Background group" on page 359).

#### Plot Area group properties

The group includes the following properties:

- "gridColor" on page 373
- "plotBgColor" on page 374
- "plotBgGradientFlag" on page 374
- "plotBgImage" on page 374

#### gridColor

To set the color of the grid lines, select the ... button and choose a color from the palette. Close the Color Chooser window when you are done.

This property is in the **Plot Area** property group.

#### plotBgColor

To set the color of the plot area, select the ... button and choose a color from the palette to set the background color. Close the Color Chooser window when you are done.

This property is in the **Plot Area** property group.

#### plotBgGradientFlag

Select to display a gradient in the plot area background. Set the color of the plot area background with the plotBgColor property.

This property is in the **Plot Area** property group.

#### plotBgImage

Specify an image (`.gif`, `.jpg`, or `.png` file) to display in the plot area. Select the name of the image file from the drop down menu, or enter the pathname of the file. The drop down menu contains the names of image files located in the current directory (typically, the `dashboards` directory of your project directory, under your Apama installation's work directory), as well as image files located in the first level of subdirectories. If you enter a pathname, use an absolute pathname or a pathname that is relative to the current directory.

This property is in the **Plot Area** property group.

## Radar graph: Radial Axis group

Properties in this group control the visibility and appearance of the radial axis, radial axis labels, and radial grid lines.

#### Radial Axis group

The group includes the following properties:

- "radialAxisColor" on page 374
- "radialAxisLabelVisFlag" on page 375
- "radialAxisLineStyle" on page 375
- "radialAxisMinLabelWidth" on page 375
- "radialAxisVisFlag" on page 375
- "radialGridLineStyle " on page 375
- "radialGridVisFlag " on page 375

#### radialAxisColor

To set the color of the radial axis and radial axis label, select the **...** button and choose a color from the palette. Close the Color Chooser window when you are done.

This property is in the **Radial Axis** property group.

#### radialAxisLabelVisFlag

Controls the visibility of the labels that appear along the radial axis.

This property is in the **Radial Axis** property group.

#### radialAxisLineStyle

Controls the style of the radial axis. Choose either **No Line**, **Solid**, **Dotted**, **Dashed**, or **Dot Dashed**.

This property is in the **Radial Axis** property group.

#### radialAxisMinLabelWidth

Specifies the minimum width in pixels for the labels that appear along the radial axis.

This property is in the **Radial Axis** property group.

#### radialAxisVisFlag

Controls the visibility of the radial axis.

This property is in the **Radial Axis** property group.

#### radialGridLineStyle

Controls the style of the radial grid lines. Choose either **No Line**, **Solid**, **Dotted**, **Dashed**, or **Dot Dashed**.

This property is in the **Radial Axis** property group.

#### radialGridVisFlag

Controls the visibility of the radial grid lines.

This property is in the **Radial Axis** property group.

## Radar graph: Trace group

Properties in this group control the appearance of trace lines and trace markers (but see also "Radar graph: Marker group" on page 371), including color, style, and line width.

#### Trace group properties

This group includes the following properties:

- "traceFillStyle" on page 376
- "traceProperties" on page 376

### traceFillStyle

Set `traceFillStyle` to one of the following fill styles for the area under the trace:

- **Solid**
- **Transparent**
- **Gradient**
- **Transparent Gradient**
- **None**

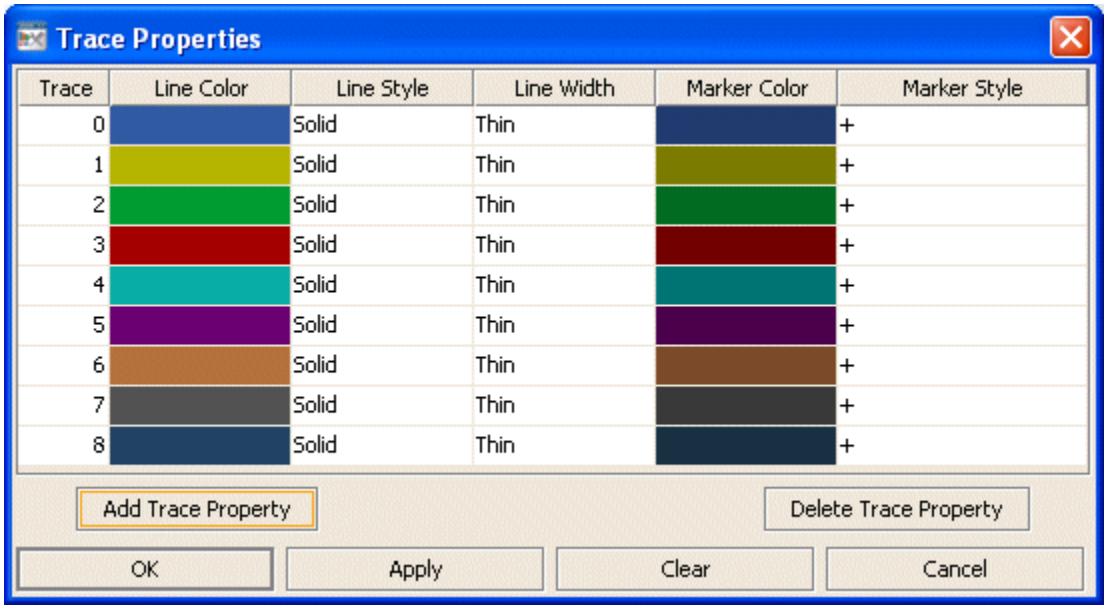
None is the default.

This property is in the **Trace** property group.

### traceProperties

Specify the line color, line style, line width, marker color and marker style of all traces.

In the Object Properties window, double-click on `traceProperties` in the **Property Name** field to bring up the Trace Properties dialog. In the Trace Properties dialog, you can assign attributes to each plotting trace in your graph.




The dialog has six columns of fields:

- **Trace**: One field for each trace that is currently in the graph. Current settings for each trace are shown.
- **Line Color**: Select the ellipsis button in the **Color** column and choose a color from the palette. Close the **Color Chooser** window.
- **Line Style**: Select the ellipsis button in the **Line Style** column and choose a style from the drop down menu. Choose either **No Line**, **Solid**, **Dotted**, **Dashed**, or **Dot Dashed**.

- **Line Width**: Select the ellipsis button in the **Line Width** column and choose a size from the drop down menu. Choose either **Thin**, **Medium** or **Thick**.
- **Marker Color**: Select the ellipsis button in the **Marker Color** column and choose a color from the palette. Close the **Color Chooser** window.
- **Marker Style**: Select the ellipsis button in the **Marker Style** column and choose a style from the drop down menu. Choose either **No Marker**, **Dot**, **+**, *****, **o**, **x**, **Filled Circle**, **Filled Diamond**, **Filled Triangle**, **Filled Square**, or **Filled Star**.

The dialog contains the following buttons:

- **Add Trace Property**: Click to add a trace property field. The data for the trace does not have to be available yet. You may consider adding and assigning attributes to more traces than your data currently needs for when you have more data to show. It is not necessary to set properties for each trace you currently or subsequently have. This is optional and can be done after additional data is displayed in a subsequent new trace.
- **Delete Trace Property**: Removes the last trace property field from the **Trace Properties** dialog.
- **OK**: Applies values and closes the dialog.
- **Apply**: Applies values without closing the dialog.
- **Reset**: Resets all fields to last values applied. Specify the line color, line style, line width, marker color and marker style of all traces.

This property is in the **Trace** property group.

## Radar graph: Value Axis group

Properties in this group control the visibility and range of the value axis, as well as value-axis label formats and value-axis divisions. They also control the visibility of value-axis grid lines.

### Value Axis group properties

The group includes the following properties:


- "valueAxisAutoScaleMode" on page 378
- "valueAxisColor" on page 378
- "valueAxisFlag" on page 378
- "valueAxisFormat" on page 378
- "valueAxisLineStyle" on page 378
- "valueAxisMajorDivisions" on page 378
- "valueAxisMinorDivisions" on page 379
- "valueGridLineStyle" on page 379

- "valueGridVisFlag" on page 379

#### valueAxisAutoScaleMode

Select one of the following modes to control the y-axis range:

- **Off**: The valueMin and valueMax properties determine the range of the value axis. This is the default. The chart origin (the bottom of the value axis) is labeled with valueMin. The intersection of the value axis and the radial axis (the top of the value axis) is labeled with valueMax.
- **On**: The dashboard calculates the value axis range according to data values being plotted.
- **On - Include Min/Max**: The dashboard calculates the smallest range that includes both valueMin and valueMax as well as all plotted points.

This property is in the **Value Axis** property group.

#### valueAxisColor

To set the color of the value axis and value axis labels, select the **...** button and choose a color from the palette. Close the Color Chooser window when you are done.

This property is in the **Value Axis** property group.

#### valueAxisFlag

Controls the visibility of the value axis.

This property is in the **Value Axis** property group.

#### valueAxisFormat

Sets the format of the numerical labels that appear along the value axis. Select or enter the format specification. Use syntax based on the Java `DecimalFormat` class.

This property is in the **Value Axis** property group.

#### valueAxisLineStyle

Controls the style of the value axis. Choose either **No Line**, **Solid**, **Dotted**, **Dashed**, or **Dot Dashed**.

This property is in the **Value Axis** property group.

#### valueAxisMajorDivisions

Specifies the number of major divisions on the value axis. Each major division is separated by a value grid line. A numeric label appears along the value axis at the intersections with the grid lines (as well as at the origin and at the intersection with the radial axis).

This property is in the **Value Axis** property group.

#### valueAxisMinorDivisions

Specifies the number of minor divisions on the value axis. Each minor division is separated by a horizontal tick mark.

This property is in the **Value Axis** property group.

#### valueGridLineStyle

Controls the style of the value grid lines. Choose either **No Line**, **Solid**, **Dotted**, **Dashed**, or **Dot Dashed**.

This property is in the **Value Axis** property group.

#### valueGridVisFlag

Controls the visibility of the value grid lines.

This property is in the **Value Axis** property group.

## XY graph

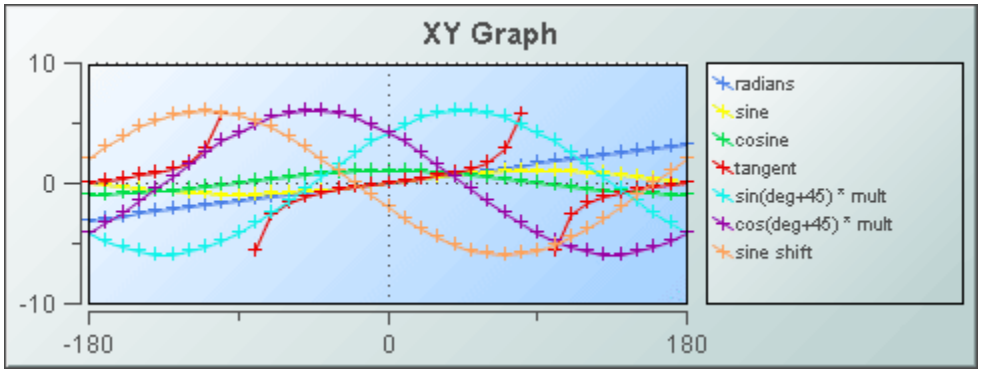
XY graphs visualize tabular data that has at least two rows with two or more numeric columns. An XY graph can visualize data in either of two ways:

- Row series visualization: The graph has one trace for each non-first row of the visualized data. (The first row is used for the x components of the plotted points, as described below.) Within each trace, there is a plotted point for each column of the data.

For a given point in a given trace, the x component is the value of the point's corresponding column for the first row of the visualized data, and the y component is the value of that column for the trace's corresponding row.

- Column series visualization: The graph has one trace for each numeric column of the visualized data, except for the first numeric column. (The first numeric column is used for the x components of the plotted points, as described below.) Within each trace, there is a plotted point for each row of the data.

For a given point in a given trace, the x component is the value of the first numeric column for the point's corresponding row, and the y component is the value of the trace's corresponding column for the point's corresponding row.




Use the valueTable property to attach data to an XY graph. Use the rowSeriesFlag property to specify row series or column series visualization.

When an XY graph is selected in the Builder canvas, the **Object Class Name** that appears at the top of the **Object Properties** pane is **obj_xygraph**.

The **Object Properties** panel organizes XY graph properties into the groups below.

## XY graph: Alert group

Properties in this group allow you to specify changes in the appearance of trace lines and markers that signal changes in the status of specified data elements. You can specify threshold values (see valueHighAlarm, valueHighWarning, valueLowAlarm, and valueLowWarning) or attach a data table to valueAlarmStatusTable that indicates the status of each element of the table that is attached to valueTable.

### Alert group propeties

This group includes the following properties:


- "valueAlarmStatusTable" on page 381
- "valueHighAlarm" on page 382
- "valueHighAlarmEnabledFlag" on page 382
- "valueHighAlarmLineVisFlag" on page 382
- "valueHighAlarmMarkColor" on page 382
- "valueHighAlarmMarkStyle" on page 382
- "valueHighAlarmTraceColor" on page 383
- "valueHighAlarmTraceStyle" on page 383
- "valueHighWarning" on page 383
- "valueHighWarningEnabledFlag" on page 383
- "valueHighWarningLineVisFlag" on page 383
- "valueHighWarningMarkColor" on page 383

- "valueHighWarningMarkStyle" on page 384
- "valueHighWarningTraceColor" on page 384
- "valueHighWarningTraceStyle" on page 384
- "valueLowAlarm" on page 384
- "valueLowAlarmEnabledFlag" on page 385
- "valueLowAlarmLineVisFlag" on page 385
- "valueLowAlarmMarkColor" on page 385
- "valueLowAlarmMarkStyle" on page 385
- "valueLowAlarmTraceColor" on page 386
- "valueLowAlarmTraceStyle" on page 386
- "valueLowWarning" on page 386
- "valueLowWarningEnabledFlag" on page 386
- "valueLowWarningLineVisFlag" on page 386
- "valueLowWarningMarkColor" on page 386
- "valueLowWarningMarkStyle" on page 387
- "valueLowWarningTraceColor" on page 387
- "valueLowWarningTraceStyle" on page 387


#### valueAlarmStatusTable

Attach an alarm table containing status indexes to this property in order to enable rule based alarm statuses for trace markers. The table attached to `valueAlarmStatusTable` must have the same number of rows and columns as valueTable. For each data element in valueTable, the status index at the corresponding position in `valueAlarmStatusTable` is used to set the alarm status of the marker that represents the data element.

Following are the valid indexes are:

- **0**: Use normal marker color and style. See "traceProperties" on page 404.
- **1**: Use low alarm marker color and style "valueLowAlarmMarkColor" on page 385 and "valueLowAlarmMarkStyle" on page 385.
- **2**: Use low warning marker color and style. See "valueLowWarningMarkColor" on page 386 and "valueLowWarningMarkStyle" on page 387.
- **3**: Use high warning marker color and style. See "valueHighWarningMarkColor" on page 383 and "valueHighWarningMarkStyle" on page 384.
- **4**: Use high alarm marker color and style. See "valueHighAlarmMarkColor" on page 382 and "valueHighAlarmMarkStyle" on page 382.

- **-1**: Determine marker color and style by comparing the value to the enabled alarm thresholds. See "valueHighAlarm" on page 382, "valueHighWarning" on page 383, "valueLowAlarm" on page 384, and "valueLowWarning" on page 386.

If no data is attached to `valueAlarmStatusTable`, the alarm status for a trace marker is determined by comparing the marker's value to the enabled thresholds. See valueHighAlarm, valueHighWarning, valueLowAlarm, and valueLowWarning.

This property is in the **Alert** property group.

#### valueHighAlarm

Specifies the threshold value used by valueHighAlarmLineVisFlag, valueHighAlarmMarkColor, valueHighAlarmMarkStyle, valueHighAlarmTraceColor, and valueHighAlarmTraceStyle.

This property is in the **Alert** property group.

#### valueHighAlarmEnabledFlag

Select to enable the high alarm threshold. See valueHighAlarm.

This property is in the **Alert** property group.

#### valueHighAlarmLineVisFlag

Select to display a dotted line at the high alarm threshold. The color of the line is set to "valueHighAlarmMarkColor" on page 382. This line is displayed only if valueHighAlarmEnabledFlag is selected.

This property is in the **Alert** property group.

#### valueHighAlarmMarkColor

When a trace marker's value is greater than or equal to valueHighAlarm, the marker changes to `valueHighAlarmMarkColor` and valueHighAlarmMarkStyle, provided valueHighAlarmEnabledFlag is selected and no data is attached to valueAlarmStatusTable.

If data is attached to valueAlarmStatusTable, a marker changes to `valueHighAlarmMarkColor` and valueHighAlarmMarkStyle when the marker's corresponding element in the attached alarm status table is 4.

If data is attached to valueAlarmStatusTable, and the marker's corresponding element in the attached alarm status table is -1, marker color and style behave as if no data were attached to valueAlarmStatusTable.

This property is in the **Alert** property group.

#### valueHighAlarmMarkStyle

When a trace marker's value is greater than or equal to valueHighAlarm, the marker changes to valueHighAlarmMarkColor and `valueHighAlarmMarkStyle`,

provided valueHighAlarmEnabledFlag is selected and no data is attached to valueAlarmStatusTable.

If data is attached to valueAlarmStatusTable, a marker changes to valueHighAlarmMarkColor and `valueHighAlarmMarkStyle` when the marker's corresponding element in the attached alarm status table is 4.

If data is attached to valueAlarmStatusTable, and the marker's corresponding element in the attached alarm status table is -1, marker color and style behave as if no data were attached to valueAlarmStatusTable.

This property is in the **Alert** property group.

#### valueHighAlarmTraceColor

When the value of any segment of a trace line is greater than or equal to valueHighAlarm, that segment of the trace line changes to `valueHighAlarmTraceColor` and valueHighAlarmTraceStyle, provided valueHighAlarmEnabledFlag is selected.

This property is in the **Alert** property group.

#### valueHighAlarmTraceStyle

When the value of any segment of a trace line is greater than or equal to valueHighAlarm, that segment of the trace line changes to valueHighAlarmTraceColor and `valueHighAlarmTraceStyle`, provided valueHighAlarmEnabledFlag is selected.

This property is in the **Alert** property group.

#### valueHighWarning

Specifies the threshold value used by valueHighWarningLineVisFlag, valueHighWarningMarkColor, valueHighWarningMarkStyle, valueHighWarningTraceColor, and valueHighWarningTraceStyle.

This property is in the **Alert** property group.

#### valueHighWarningEnabledFlag

Select to enable the high warning threshold. See valueHighWarning.

This property is in the **Alert** property group.

#### valueHighWarningLineVisFlag

Select to display a dotted line at the high warning threshold. The color of the line is set to valueHighWarningMarkColor. This line is displayed only if valueHighWarningEnabledFlag is selected.

This property is in the **Alert** property group.

#### valueHighWarningMarkColor

When a trace marker's value is greater than or equal to valueHighWarning but less than valueHighAlarm, the marker changes to `valueHighWarningMarkColor` and

valueHighWarningMarkStyle, provided valueHighWarningEnabledFlag is selected and no data is attached to valueAlarmStatusTable.

If data is attached to valueAlarmStatusTable, a marker changes to `valueHighWarningMarkColor` and valueHighWarningMarkStyle when the marker's corresponding element in the attached alarm status table is 3.

If data is attached to valueAlarmStatusTable, and the marker's corresponding element in the attached alarm status table is -1, marker color and style behave as of no data were attached to valueAlarmStatusTable.

This property is in the **Alert** property group.

#### valueHighWarningMarkStyle

When a trace marker's value is greater than or equal to valueHighWarning but less than valueHighAlarm, the marker changes to valueHighWarningMarkColor and `valueHighWarningMarkStyle`, provided valueHighWarningEnabledFlag is selected and no data is attached to valueAlarmStatusTable.

If data is attached to valueAlarmStatusTable, a marker changes to valueHighWarningMarkColor and `valueHighWarningMarkStyle` when the marker's corresponding element in the attached alarm status table is 3.

If data is attached to valueAlarmStatusTable, and the marker's corresponding element in the attached alarm status table is -1, marker color and style behave as of no data were attached to valueAlarmStatusTable.

This property is in the **Alert** property group.

#### valueHighWarningTraceColor

When the value of any segment of a trace line is greater than or equal to valueHighWarning property but less than valueHighAlarm, that segment of the trace line changes to `valueHighWarningTraceColor` and valueHighWarningTraceStyle, provided valueHighWarningEnabledFlag is selected.

This property is in the **Alert** property group.

#### valueHighWarningTraceStyle

When the value of any segment of a trace line is greater than or equal to valueHighWarning property but less than valueHighAlarm, that segment of the trace line changes to valueHighWarningTraceColor and `valueHighWarningTraceStyle`, provided valueHighWarningEnabledFlag is selected.

This property is in the **Alert** property group.

#### valueLowAlarm

Specifies the threshold value used by valueLowAlarmLineVisFlag, valueLowAlarmMarkColor, valueLowAlarmMarkStyle, valueLowAlarmTraceColor, and valueLowAlarmTraceStyle.

This property is in the **Alert** property group.

#### valueLowAlarmEnabledFlag

Select to enable the low alarm threshold. See valueLowAlarm.

This property is in the **Alert** property group.

#### valueLowAlarmLineVisFlag

Select to display a dotted line at the low alarm threshold. The color of the line is set to valueLowAlarmMarkColor. This line is displayed only if valueLowAlarmEnabledFlag is selected.

This property is in the **Alert** property group.

#### valueLowAlarmMarkColor

When a trace marker's value is less than or equal to valueLowAlarm, the marker changes to `valueLowAlarmMarkColor` and valueLowAlarmMarkStyle, provided valueLowAlarmEnabledFlag is selected and no data is attached to valueAlarmStatusTable.

If data is attached to valueAlarmStatusTable, a marker changes to `valueLowAlarmMarkColor` and valueLowAlarmMarkStyle when the marker's corresponding element in the attached alarm status table is 1.

If data is attached to valueAlarmStatusTable, and the marker's corresponding element in the attached alarm status table is -1, marker color and style behave as of no data were attached to valueAlarmStatusTable.

This property is in the **Alert** property group.

#### valueLowAlarmMarkStyle

When a trace marker's value is less than or equal to valueLowAlarm, the marker changes to valueLowAlarmMarkColor and `valueLowAlarmMarkStyle`, provided valueLowAlarmEnabledFlag is selected and no data is attached to valueAlarmStatusTable.

If data is attached to valueAlarmStatusTable, a marker changes to valueLowAlarmMarkColor and `valueLowAlarmMarkStyle` when the marker's corresponding element in the attached alarm status table is 1.

If data is attached to valueAlarmStatusTable, and the marker's corresponding element in the attached alarm status table is -1, marker color and style behave as of no data were attached to valueAlarmStatusTable.

This property is in the **Alert** property group.

#### valueLowAlarmTraceColor

When the value of any segment of a trace line is less than or equal to valueLowAlarm, that segment of the trace line changes to `valueLowAlarmTraceColor` and valueLowAlarmTraceStyle, provided valueLowAlarmEnabledFlag is selected.

This property is in the **Alert** property group.

#### valueLowAlarmTraceStyle

When the value of any segment of a trace line is less than or equal to valueLowAlarm, that segment of the trace line changes to valueLowAlarmTraceColor and `valueLowAlarmTraceStyle`, provided valueLowAlarmEnabledFlag is selected.

This property is in the **Alert** property group.

#### valueLowWarning

Specifies the threshold value used by valueLowWarningLineVisFlag, valueLowAlarmMarkColor, valueLowWarningMarkStyle, valueLowWarningTraceColor, and valueLowWarningTraceStyle.

This property is in the **Alert** property group.

#### valueLowWarningEnabledFlag

Select to enable the low warning threshold. See valueLowWarning.

This property is in the **Alert** property group.

#### valueLowWarningLineVisFlag

Select to display a dotted line at the low warning threshold. The color of the line is set to valueLowWarningMarkColor. This line is displayed only if valueLowWarningEnabledFlag is selected.

This property is in the **Alert** property group.

#### valueLowWarningMarkColor

When a trace marker's value is less than or equal to valueLowWarning but greater than valueLowAlarm, the marker changes to `valueLowWarningMarkColor` and valueLowWarningMarkStyle, provided valueLowWarningEnabledFlag is selected and no data is attached to valueAlarmStatusTable.

If data is attached to valueAlarmStatusTable, a marker changes to `valueLowWarningMarkColor` and valueLowWarningMarkStyle when the marker's corresponding element in the attached alarm status table is 2.

If data is attached to valueAlarmStatusTable, and the marker's corresponding element in the attached alarm status table is -1, marker color and style behave as of no data were attached to valueAlarmStatusTable.

This property is in the **Alert** property group.

#### valueLowWarningMarkStyle

When a trace marker's value is less than or equal to valueLowWarning but greater than valueLowAlarm, the marker changes to valueLowWarningMarkColor and `valueLowWarningMarkStyle`, provided valueLowWarningEnabledFlag is selected and no data is attached to valueAlarmStatusTable.

If data is attached to valueAlarmStatusTable, a marker changes to valueLowWarningMarkColor and `valueLowWarningMarkStyle` when the marker's corresponding element in the attached alarm status table is 2.

If data is attached to valueAlarmStatusTable, and the marker's corresponding element in the attached alarm status table is -1, marker color and style behave as of no data were attached to valueAlarmStatusTable.

This property is in the **Alert** property group.

#### valueLowWarningTraceColor

When the value of any segment of a trace line is less than or equal to valueLowWarning but greater than valueLowAlarm, that segment of the trace line changes to `valueLowWarningTraceColor` and valueLowWarningTraceStyle, provided valueLowWarningEnabledFlag is selected.

This property is in the **Alert** property group.

#### valueLowWarningTraceStyle

When the value of any segment of a trace line is less than or equal to valueLowWarning property but greater than valueLowAlarm, that segment of the trace line changes to valueLowWarningTraceColor and `valueLowWarningTraceStyle`, provided valueLowWarningEnabledFlag is selected.

This property is in the **Alert** property group.

## XY graph: Background group

Properties in this group control the visibility and appearance of the portion of the graph that serves as the background of both the plot area and legend.

#### Background group properties

The group contains the following properties:


- "bgBorderColor" on page 388
- "bgBorderFlag" on page 388
- "bgColor" on page 388
- "bgEdgeWidth" on page 388
- "bgGradientColor2" on page 388

- "bgGradientMode" on page 388
- "bgRaisedFlag" on page 389
- "bgRoundness" on page 389
- "bgShadowFlag" on page 389
- "bgStyleFlag" on page 389
- "bgVisFlag" on page 390
- "borderPixels" on page 390


#### bgBorderColor

Sets the color of the border (see `bgBorderFlag`) of the background rectangle. Select the **...** button and choose a color from the palette. Close the Color Chooser window when you are done.

#### bgBorderFlag

Select to display a border around the background rectangle.

This property is in the **Background** property group.

#### bgColor

Sets the background color. Select the **...** button and choose a color from the palette. Close the Color Chooser window when you are done.

This property is in the **Background** property group.

#### bgEdgeWidth

Sets the width in pixels of the 3D edge on the background rectangle. This property is only used if `bgBorderFlag` is selected.

This property is in the **Background** property group.

#### bgGradientColor2

Sets the color for the second color in the gradient. The default is white. The `bgColor` property sets the first color in the gradient.

This property is in the **Background** property group.

#### bgGradientMode

Display a gradient in the background rectangle. Select from the following options:

- **None**: No gradient
- **Diagonal Edge**: Gradient is drawn at a 45 degree angle from the top left to the bottom right corner of the object.

- **Diagonal Center**: Gradient is drawn at a 45 degree angle from the center to the top left and the bottom right corners of the object.
- **Horizontal Edge**: Gradient is drawn horizontally from the top to the bottom of the object.
- **Horizontal Center**: Gradient is drawn horizontally from the center to the top and bottom of the object.
- **Vertical Edge**: Gradient is drawn vertically from the left to the right of the object.
- **Vertical Center**: Gradient is drawn vertically from the center to the left and right of the object.

This property is in the **Background** property group.

#### bgRaisedFlag

Reverses the direction of the gradient, as well as that of the 3D edge if the `bgStyle` selected is 3D Rectangle.

This property is in the **Background** property group.

#### bgRoundness

Sets the arc length of the rounded corners. This property is only available if the `bgStyle` selected is **Round Rectangle**.

The value of `bgRoundness` cannot exceed half the value of the `objWidth` or the `objHeight`. If `bgRoundness` does exceed that value, half of `objWidth` or `objHeight` (whichever is smaller) will be used instead. For example if `objWidth` is `100` and `objHeight` is `50`, then the value of `bgRoundness` cannot exceed `25`. If it does, then half the value of `objHeight` (`25`) will be used instead. This property is in the **Background** property group.

#### bgShadowFlag

Select to display a drop shadow on the background rectangle.

This property is in the **Background** property group.

#### bgStyleFlag

Choose one of the following three options from the drop down menu:

- **Rectangle**: Select to display a background rectangle.
- **3D Rectangle**: Select to display a 3D edge on the background rectangle. If selected, use `bgEdgeWidth` to set the width of the 3D edge.
- **Round Rectangle**: Select to display a background rectangle with rounded edges. If selected, use `bgRoundness` to set the arc length of the rounded corners.

This property is in the **Background** property group.

#### bgVisFlag

Select to display the background rectangle.

This property is in the **Background** property group.

#### borderPixels

Sets the width in pixels of the border between the chart and the edge of the background rectangle.

This property is in the **Background** property group.

## XY graph: Column group

This group contains one property, columnsToHide , which controls which data-attachment columns are excluded from being used for plotted data or labels.

#### columnsToHide

Specifies columns from the data attachment to exclude from being used for plotted data or labels. Data from the labelColumnName column are used for labels even if that column name is also specified in the `columnsToHide` property. Columns specified in the `columnsToHide` property can still be used in the drillDownColumnSubs property.

This property is in the **Column** property group.

## XY graph: Data group

Properties in this group control what data appears in the graph, as well as whether the data appears in column series or row series form.

#### Data group properties

The group contains the following properties:


- "rowSeriesFlag" on page 390
- "valueTable" on page 391
- "xValueDivisor" on page 391
- "xValueMax" on page 392
- "xValueMin" on page 392
- "yValueDivisor" on page 392
- "yValueMax" on page 392
- "yValueMin" on page 392


#### rowSeriesFlag

Controls how x and y data populate the graph:

- If the `rowSeriesFlag` checkbox is selected, the graph has one trace for each non-first row of your data attachment. (The first row is used for the x components of the plotted points, as described below.) Within each trace, there is a plotted point for each column of your attachment.

For a given point in a given trace, the x component is the value of the point's corresponding column for the first row of your attachment, and the y component is the value of that column for the trace's corresponding row. Values from the label column (see labelColumnName) or generated row identifiers are used as labels in the legend.

- If the `rowSeriesFlag` checkbox is not selected, there is a trace for each numeric column of your data attachment, except for the first numeric column. (The first numeric column is used for the x components of the plotted points, as described below.) Within each trace, there is a plotted point for each row of your attachment.

For a given point in a given trace, the x component is the value of the first numeric column for the point's corresponding row, and the y component is the value of the trace's corresponding column for the point's corresponding row. Column names appear in the legend.

This property is in the **Data** property group.

#### valueTable

Attach your data to the `valueTable` property. Your data attachment must contain at least two rows and at least two numeric columns. The property rowSeriesFlag controls how x and y data populate the graph:

- If the rowSeriesFlag checkbox is selected, the graph has one trace for each non-first row of your data attachment. (The first row is used for the x components of the plotted points, as described below.) Within each trace, there is a plotted point for each column of your attachment.

For a given point in a given trace, the x component is the value of the point's corresponding column for the first row of your attachment, and the y component is the value of that column for the trace's corresponding row. Values from the label column (see labelColumnName) or generated row identifiers are used as labels in the legend.

- If the rowSeriesFlag checkbox is not selected, there is a trace for each numeric column of your data attachment, except for the first numeric column. (The first numeric column is used for the x components of the plotted points, as described below.) Within each trace, there is a plotted point for each row of your attachment.

For a given point in a given trace, the x component is the value of the first numeric column for the point's corresponding row, and the y component is the value of the trace's corresponding column for the point's corresponding row. Column names appear in the legend.

This property is in the **Data** property group.

#### xValueDivisor

The x values are divided by the value entered into the `xValueDivisor`. The default is 1.

This property is in the **Data** property group.

#### xValueMax

The xValueMin and `xValueMax` properties control the range of the x-axis if xAxisAutoScaleMode is set to Off. In addition, if xAxisAutoScaleMode is set to **On - Include Min/Max**, the dashboard calculates the smallest x-axis range that includes both xValueMin and `xValueMax` as well as all plotted points.

This property is in the **Data** property group.

#### xValueMin

The `xValueMin` and xValueMax properties control the range of the x-axis if xAxisAutoScaleMode is set to Off. In addition, if xAxisAutoScaleMode is set to **On - Include Min/Max**, the dashboard calculates the smallest x-axis range that includes both `xValueMin` and xValueMax as well as all plotted points.

This property is in the **Data** property group.

#### yValueDivisor

The y values are divided by the value entered into the `yValueDivisor`. The default is 1.

This property is in the **Data** property group.

#### yValueMax

The yValueMin and `yValueMax` properties control the range of the y-axis if the yAxisAutoScaleMode is set to Off. In addition, if yAxisAutoScaleMode is set to **On - Include Min/Max**, the dashboard calculates the smallest y-axis range that includes both yValueMin and `yValueMax` as well as all plotted points.

This property is in the **Data** property group.

#### yValueMin

The `yValueMin` and yValueMax properties control the range of the y-axis if the yAxisAutoScaleMode is set to Off. In addition, if yAxisAutoScaleMode is set to **On - Include Min/Max**, the dashboard calculates the smallest y-axis range that includes both `yValueMin` and yValueMax as well as all plotted points.

This property is in the **Data** property group.

## XY graph: Data Format group

Properties on this group control the format of displayed values as well as numerical and date labels.

#### Data Format group properties

The group includes the following properties:

- "labelColumnFormat" on page 393
- "xValueFormat" on page 393
- "yValueFormat" on page 393


#### labelColumnFormat

Select or enter the format of numeric or date labels displayed in the legend and popup legend (see legendPopupFlag).

For numeric labels, use syntax from the Java `DecimalFormat` class.

For date labels, use from the Java `SimpleDateFormat` class.

This property is in the **Data Format** property group.

#### xValueFormat

Sets the numeric format of trace values displayed in the legend and popup legend.

Select or enter a format. Use syntax from the Java `DecimalFormat` class.

This property is in the **Data Format** property group.

#### yValueFormat

Sets the numeric format of trace values displayed in the legend and popup legend.

Select or enter a format. Use syntax from the Java `DecimalFormat` class.

This property is in the **Data Format** property group.

## XY graph: Data Label group

Properties in this group control the labels that are used in the legend.

#### Data Label group properties

The group contains the following properties:


- "columnDisplayNames" on page 393
- "labelColumnName" on page 394
- "rowLabelVisFlag" on page 394
- "rowNameVisFlag" on page 394


#### columnDisplayNames

Sets alternate display names for column names in your XY graph's data. Column names are displayed in the legend when rowSeriesFlag is not selected.

This property is in the **Data Label** property group.

#### labelColumnName

Sets the label column. By default, the label column is the first non-numeric text column in your data attachment, if there is one (for Apama data tables, **apama.instanceID** is used if there is no other non-numeric column).

Data from the label column is used to label the legend, if both rowLabelVisFlag and rowSeriesFlag are enabled.

This property is in the **Data Label** property group.

#### rowLabelVisFlag

Determines whether data from the label column (see labelColumnName) is used for legend labels. Data from the label column is used to label the legend if both `rowLabelVisFlag` and rowSeriesFlag are enabled.

This property is in the **Data Label** property group.

#### rowNameVisFlag

If your data attachment has no label column (see labelColumnName), select this property to use generated row names in the legend when the rowSeriesFlag is not selected.

This property is in the **Data Label** property group.

## XY graph: Historian group

Do not use the properties in this group.

#### historyTableName

Do not use this property.

This property is in the **Historian** property group.

#### historyTableRowNameFlag

Do not use this property.

This property is in the **Historian** property group.

## XY graph: Interaction group

Properties in this group control various forms of interaction between the end user and the graph, including scrolling, zooming, and activating commands, drill downs, and tooltips.

#### Interaction group properties

The group includes the following properties:

- "command" on page 395
- "commandCloseWindowOnSuccess" on page 396
- "commandConfirm" on page 396
- "confirmText" on page 396
- "cursorColor" on page 396
- "cursorFlag" on page 396
- "drillDownColumnSubs" on page 397
- "drillDownSelectMode" on page 397
- "drillDownTarget" on page 397
- "scrollbarMode" on page 397
- "scrollbarSize" on page 398
- "mouseOverFlag" on page 398
- "zoomEnabledFlag" on page 398


**command**

Assign a command or group of commands to this stock chart by right-clicking on the `command` property name in the Object Properties window. Select **Define Command** and choose **SYSTEM**, **APAMA**, or **MULTIPLE**. See "Using the Define Apama Command dialog" on page 226.

Once a command or command group has been assigned to this object, you can activate it from a deployed dashboard or from the Dashboard Builder:

- Dashboard Builder: Double click on the object.
- Web-based deployment: Single click on the object or else right click on it and select **Execute Command** from the popup menu.
- Local deployment: By default, single-click on the object or else right-click on it and select **Execute Command** from the popup menu. To override the default, select **Tools > Options** in the Builder (do this before you generate the deployment package), and uncheck **Single-Click for Drill Down and Commands** in the **General** tab. This allows the end user to use either a double click or a right click.

When you activate a command, any defined drill down substitutions are performed, and then the command is executed.

If you assign multiple commands, the commands are launched in an arbitrary order, and are executed asynchronously; there is no guarantee that one command will finish before the next one in the sequence starts.

This property is in the **Interaction** property group.

#### commandCloseWindowOnSuccess

Select this property to automatically close the window that initiates a **SYSTEM** command when the command is executed successfully. This applies to **SYSTEM** commands only, and is not supported at all for thin-client, Web-page deployments.

With **APAMA** commands, the window is closed whether or not the command is executed successfully. For **MULTIPLE** commands, the window closes when the first command in the command group succeeds.

This property is in the **Interaction** property group.

#### commandConfirm

By default, when the end user executes a command (see the `command` property), the command confirmation dialog is disabled. To control this option for each individual object, use the `commandConfirm` check box. If confirmation is required for a **MULTIPLE** command group, a single confirmation dialog is presented; if you confirm the execution, all individual commands in the group are executed with no further confirmation. If the you cancel the execution, none of the commands in the group is executed.

You can also override the confirmation status of individual objects with an application-wide policy. Select **Tools > Options** and choose from three confirmation values:

- **Do not confirm**: Indicates that no commands require confirmation (regardless of each object's confirmation status).
- **Confirm all**: Indicates that all commands require confirmation (regardless of each object's confirmation status).
- **Use object confirm flag** (default): Indicates that the confirmation status of each object will determine whether confirmation is required.

This property is in the **Interaction** property group.

#### confirmText

Use this property to write your own text for the confirmation dialog. Otherwise, default text is used. See commandConfirm.

This property is in the **Interaction** property group.

#### cursorColor

To set the color of the cursor (see cursorFlag), select the **...** button and choose a color from the palette. Close the Color Chooser window when you are done.

This property is in the **Interaction** property group.

#### cursorFlag

Select to enable the cursor. When the cursor is enabled, point to a location on a trace to see a cursor line at that location and display the time and values of all traces at the

cursor line on the legend. Hold down the control key to snap the cursor to the closest data point. Select the legendPopupFlag to display the legend along the cursor.

This property is in the **Interaction** property group.

#### drillDownColumnSubs

Use this property to direct a dashboard to assign data-table column values to specified dashboard variables when the end user activates a drilldown on this object. In the Object Properties window, double-click on `drillDownColumnSubs` in the **Property Name** field to bring up the Drill Down Column Substitutions dialog.

The dialog has the following fields and buttons:

- **Substitution String**: Enter the dashboard variable next to the name of the data table column whose value you want assigned to the variable. Press **Enter**.
- **Add Column**: Enter the name of a column and click the **Add Column** button to insert a column into the table.
- **Clear**: Click the **Clear** button to remove all variables listed.

The **Column Name** list is populated based on the table's data attachment. If you have not yet attached the table to data, this list is empty.

Once you have selected which column values to pass in as substitutions, double-click on any element in your object to open a drill down window that displays corresponding values.

This property is in the **Interaction** property group.

#### drillDownSelectMode

Use this property to control how a drill down display is activated. Select one of the following:

- **Anywhere** to activate a drill down display by double-clicking anywhere on the chart.
- **Element Only** to enable a drill down display only when you double-click on an element of the chart, such as a bar or candlestick.

This property is in the **Interaction** property group.

#### drillDownTarget

To specify a drill down display, double click on `drillDownTarget` in the **Property Name** field to bring up the Drill Down Properties dialog. See "Drill-Down Specification" on page 535.

This property is in the **Interaction** property group.

#### scrollbarMode

Select one of the following from the `scrollbarMode` property to set the behavior of the scroll bar in the table:

- **Never**: Default setting
- **Always**: Display a scroll bar at all times.
- **As Needed**: Display the scroll bar when necessitated by zooming in the trace area or when you have X or Y values that are outside of the min/max range.

This property is in the **Interaction** property group.

#### scrollbarSize

Specifies the height of the horizontal scroll bar and the width of the vertical scroll bar, in pixels. The default value is -1, which sets the size to the system default.

This property is in the **Interaction** property group.

#### mouseOverFlag

Select to enable tooltips for your graph. To display a tooltip, point to a trace marker with your mouse. The tooltip contains information from your data attachment about that trace marker. This property applies only if legendPopupFlag is disabled.

This property is in the **Interaction** property group.

#### zoomEnabledFlag

Select to enable zooming within the graph. Click in the graph's trace area and drag the cursor until a desired range is selected. While dragging, a rectangle is drawn to show the zoom area. The rectangle's default color is yellow (this can be changed in the cursorColor property). After the zoom is performed, the graph stores up to four zoom operations in queue. To zoom out, press the Shift key and click in the graph's trace area.

This property is in the **Interaction** property group.

## XY graph: Label group

Properties in this group control the graph's main label (which defaults to **XY Graph**), including text, alignment, color, font, and size.

#### Label group properties

The group includes the following properties:


- "label" on page 399
- "labelTextAlignX" on page 399
- "labelTextColor" on page 399
- "labelTextFont" on page 399
- "labelTextHeight" on page 399

#### label

Specifies the text for the chart label. Click the ellipsis for multi-line text.

The default is **XY Graph**.

This property is in the **Label** property group.

#### labelTextAlignX

Sets the alignment of the chart label (see the label property). Select **Left**, **Center**, or **Right** from the drop down list.

This property is in the **Label** property group.

#### labelTextColor

Specifies the color of the chart label text (see the label property). Select the ... button and choose a color from the palette. Close the **Color Chooser** window when you are done.

This property is in the **Label** property group.

#### labelTextFont

Specifies the font of the chart label text (see the label property). Select an item from drop down list.

This property is in the **Label** property group.

#### labelTextHeight

Specifies the point size of the chart label text (see the label property).

This property is in the **Label** property group.

## XY graph: Legend group

Properties in this group control the visibility and appearance of the graph legend.

#### Legend group properties

The group contains the following properties:

- "legendBgColor" on page 400
- "legendBgGradientFlag" on page 400
- "legendPopupFlag" on page 400
- "legendValueMinSpace" on page 400
- "legendValueVisFlag" on page 400
- "legendVisFlag" on page 400
- "legendWidthPercent" on page 400

#### legendBgColor

Select the **...** button and choose a color from the palette to set the background color of the legend. Close the Color Chooser window when you are done.

This property is in the **Legend** property group.

#### legendBgGradientFlag

Select to display a gradient in the legend background.

This property is in the **Legend** property group.

#### legendPopupFlag

Select to display the legend along the cursor.

This property is in the **Legend** property group.

#### legendValueMinSpace

Specifies the minimum number of pixels between values and labels in the legend. This property applies only if legendValueVisFlag is enabled.

This property is in the **Legend** property group.

#### legendValueVisFlag

Select to display the numerical values of your data in the legend.

This property is in the **Legend** property group.

#### legendVisFlag

Select to display the legend.

This property is in the **Legend** property group.

#### legendWidthPercent

Set the percent of the total width of the object used for the legend.

This property is in the **Legend** property group.

## XY graph: Marker group

Properties in this group control the appearance of trace markers (but see also "XY graph: Trace group" on page 404).

#### Marker group properties

The group contains the following properties:

- "markDefaultSize" on page 401

- "markScaleMode" on page 401

#### markDefaultSize

Sets the size of the markers in pixels. Supply an integer value that is between 1 and 18, inclusive.

This property is in the **Marker** property group.

#### markScaleMode

Sets the scale mode for trace marks. Select one of the following from the drop down menu:

- **No Scale**: All marks, across and within traces, are the same size.
- **Scale by Trace**: Scale marks according to the trace in which they reside, that is, marks in the first trace are the largest, across all traces, and the marks in the last trace are the smallest.
- **Scale Within Trace**: Scale marks according to the relative order of the data within each trace.

This property is in the **Marker** property group.

## XY graph: Object group

Properties in this group control the visibility and transparency of the graph as a whole. They also control (or reflect) the overall position and dimensions of the graph. In addition, a property in this group reflects the generated name of this individual graph.

#### Object group properties

This group contains the following properties:

- "anchor" on page 402
- "dock" on page 402
- "objHeight" on page 402
- "objName" on page 402
- "objWidth" on page 402
- "objX" on page 402
- "objY" on page 402
- "transparencyPercent" on page 403
- "visFlag" on page 403

#### anchor

Select zero or more of **Top**, **Left**, **Bottom**, and **Right** in order to control the object's placement. The `anchor` property is only applied when the display is resized either by changing the **Background Properties** on the display or by resizing the window in **Layout** mode. If an object has the `dock` property set, the `anchor` property is ignored. See "About resize modes" on page 39.

#### dock

Select **None** (default), **Top**, **Left**, **Bottom**, **Right**, or **Fill** in order to control the object's placement in **Layout** resize mode. See "About resize modes" on page 39.

#### objHeight

Set the height of a chart by entering a value for this property or by dragging a handle of the bounding box that appears when the chart is selected. When you drag a handle of the bounding box, the displayed value for this property changes to reflect the real-time height of the chart.

This property is in the **Object** property group.

#### objName

An identifier that is generated by the Dashboard Builder. This name can be used by other objects' properties in order to refer to the named chart.

This property is in the **Object** property group.

#### objWidth

Set the width of a chart by entering a value for this property or by dragging a handle of the bounding box that appears when the chart is selected. When you drag a handle of the bounding box, the displayed value for this property changes to reflect the real-time width of the chart.

This property is in the **Object** property group.

#### objX

Sets the X coordinate of the center of this visualization object, relative to the lower left corner of the current dashboard. This value is set automatically when you position the object with the mouse.

This property is in the **Object** property group.

#### objY

Sets the Y coordinate of the center of this visualization object, relative to the lower left corner of the current dashboard. This value is set automatically when you position the object with the mouse.

This property is in the **Object** property group.

#### transparencyPercent

Sets the transparency of this chart.

This property is in the **Object** property group.

#### visFlag

Deselect to make this visualization object invisible in the current dashboard.

This property is in the **Object** property group.

## XY graph: Plot Area group

Properties in this group control the appearance of the plot area, the rectangular area that serves as background for the grid lines and trace lines (but not for the legend or axis labels; see "XY graph: Background group" on page 387).

#### Plot Area group properties

The group includes the following properties:

- "gridBgColor" on page 403
- "gridBgGradientFlag" on page 403
- "gridBgImage" on page 403
- "gridColor" on page 404

#### gridBgColor

To set the color of the plot area, select the **...** button and choose a color from the palette to set the background color. Close the Color Chooser window when you are done.

This property is in the **Plot Area** property group.

#### gridBgGradientFlag

Select to display a gradient in the grid background. Set the color of the grid background with the `gridBgColor` property.

This property is in the **Plot Area** property group.

#### gridBgImage

Specify an image (`.gif`, `.jpg`, or `.png` file) to display in the plot area. Select the name of the image file from the drop down menu, or enter the pathname of the file. The drop down menu contains the names of image files located in the current directory (by default, the `dashboards` directory of your Apama installation's work directory), as well as image files located in the first level of subdirectories. If you enter a pathname, use an absolute pathname or a pathname that is relative to the current directory.

This property is in the **Plot Area** property group.

#### gridColor

Sets the color of the dotted, horizontal midline of the plot area. Select the **...** button and choose a color from the palette. Close the Color Chooser window when you are done.

This property is in the **Plot Area** property group.

## XY graph: Trace group

Properties in this group control the appearance of trace lines and trace markers (but see also "XY graph: Marker group" on page 400), including color, style, and line width.

#### Trace group properties

This group includes the following properties:

- "traceFillStyle" on page 404
- "traceProperties" on page 404

#### traceFillStyle

Set `traceFillStyle` to one of the following fill styles for the area under the trace:

- **Solid**
- **Transparent**
- **Gradient**
- **Transparent Gradient**
- **None**

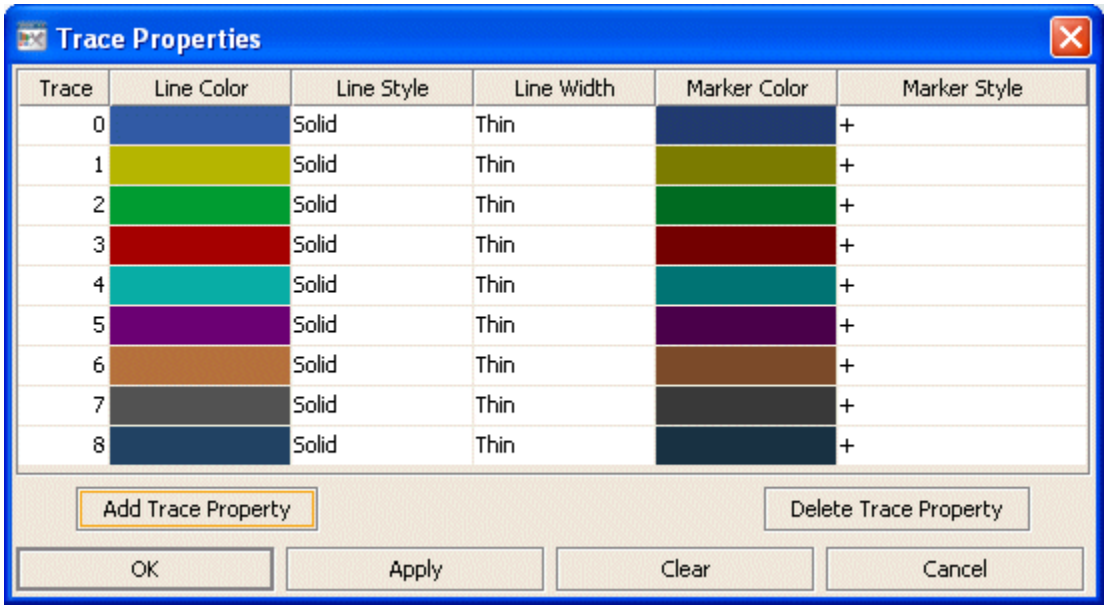
None is the default.

This property is in the **Trace** property group.

#### traceProperties

Specify the line color, line style, line width, marker color and marker style of all traces.

In the Object Properties window, double-click on `traceProperties` in the **Property Name** field to bring up the Trace Properties dialog. In the Trace Properties dialog you can assign attributes to each plotting trace in your graph.




The dialog has six columns of fields:

- **Trace**: One field for each trace that is currently in the graph. Current settings for each trace are shown.
- **Line Color**: Select the ellipsis button in the **Color** column and choose a color from the palette. Close the **Color Chooser** window.
- **Line Style**: Select the ellipsis button in the **Line Style** column and choose a style from the drop down menu. Choose either **No Line**, **Solid**, **Dotted**, **Dashed**, or **Dot Dashed**.
- **Line Width**: Select the ellipsis button in the **Line Width** column and choose a size from the drop down menu. Choose either **Thin**, **Medium** or **Thick**.
- **Marker Color**: Select the ellipsis button in the **Marker Color** column and choose a color from the palette. Close the **Color Chooser** window.
- **Marker Style**: Select the ellipsis button in the **Marker Style** column and choose a style from the drop down menu. Choose either **No Marker**, **Dot**, **+**, *****, **o**, **x**, **Filled Circle**, **Filled Diamond**, **Filled Triangle**, **Filled Square**, or **Filled Star**.

The dialog contains the following buttons:

- **Add Trace Property**: Click to add a trace property field. The data for the trace does not have to be available yet. You may consider adding and assigning attributes to more traces than your data currently needs for when you have more data to show. It is not necessary to set properties for each trace you currently or subsequently have. This is optional and can be done after additional data is displayed in a subsequent new trace.
- **Delete Trace Property**: Removes the last trace property field from the Trace Properties dialog.
- **OK**: Applies values and closes the dialog.

- **Apply**: Applies values without closing the dialog.
- **Reset**: Resets all fields to last values applied. Specify the line color, line style, line width, marker color and marker style of all traces.

This property is in the **Trace** property group.

## XY graph: X-Axis group

Properties in this group control the visibility and scaling of the x-axis, as well as x-axis label formats and x-axis divisions. They also control x-axis sorting and reversing.

### X-Axis group properties

The group includes the following properties:


- "xAxisAutoScaleMode" on page 406
- "xAxisAutoScaleRoundFlag" on page 406
- "xAxisFlag" on page 407
- "xAxisFormat" on page 407
- "xAxisLabel" on page 407
- "xAxisLabelTextHeight" on page 407
- "xAxisMajorDivisions" on page 407
- "xAxisMinorDivisions" on page 407
- "xAxisReverseFlag" on page 407
- "xValueSortFlag" on page 407


### xAxisAutoScaleMode

Select one of the following modes to control the x-axis range:

- **Off**: The xValueMin and xValueMax properties determine the range of the x-axis. This is the default.
- **On**: The dashboard calculates the x-axis range according to data values being plotted.
- **On - Include Min/Max**: The dashboard calculates the smallest range that includes xValueMin and xValueMax as well as all plotted points.

This property is in the **X-Axis** property group.

### xAxisAutoScaleRoundFlag

Select to round values on the x-axis.

This property is in the **X-Axis** property group.

#### xAxisFlag

Select to display the x-axis.

This property is in the **X-Axis** property group.

#### xAxisFormat

Sets the numeric format of values displayed on the x-axis. Select or enter a format. Use syntax from the Java `DecimalFormat` class.

This property is in the **X-Axis** property group.

#### xAxisLabel

Specifies the x-axis label.

This property is in the **X-Axis** property group.

#### xAxisLabelTextHeight

Specifies the height in pixels of the x-axis labels.

This property is in the **X-Axis** property group.

#### xAxisMajorDivisions

Specify the number of major divisions on the x-axis.

This property is in the **X-Axis** property group.

#### xAxisMinorDivisions

Specify the number of minor divisions on the x- axis.

This property is in the **X-Axis** property group.

#### xAxisReverseFlag

Select to reverse the order of the x-axis values and plot values decreasing from left to right.

This property is in the **X-Axis** property group.

#### xValueSortFlag

Select to sort data from lowest to highest x values.

This property is in the **X-Axis** property group.

## XY graph: Y-Axis group

Properties in this group control the visibility and scaling of the y-axis, as well as y-axis label formats and y-axis divisions. They also control whether there is a single y-axis, or one per trace.

#### Y-Axis group properties

The group includes the following properties:


- "yAxisAutoScaleMode" on page 408
- "yAxisFlag" on page 408
- "yAxisFormat" on page 408
- "yAxisLabel" on page 408
- "yAxisLabelTextHeight" on page 409
- "yAxisMajorDivisions" on page 409
- "yAxisMinLabelWidth" on page 409
- "yAxisMinorDivisions" on page 409
- "yAxisMultiRangeFlag" on page 409


#### yAxisAutoScaleMode

Select one of the following modes to control the y-axis range:

- **Off**: The yValueMin and yValueMax properties determine the range of the y-axis. This is the default.
- **On**: The dashboard calculates the y-axis range according to data values being plotted.
- **On - Include Min/Max**: The dashboard calculates the smallest range (with rounding) that includes both yValueMin and yValueMax as well as all plotted points.

This property is in the **Y-Axis** property group.

#### yAxisFlag

Select to display the y-axis.

This property is in the **Y-Axis** property group.

#### yAxisFormat

Sets the numeric format of values displayed on the y-axis.

Select or enter a format. Use syntax from the Java `DecimalFormat` class.

This property is in the **Y-Axis** property group.

#### yAxisLabel

Specifies the y-axis label.

This property is in the **Y-Axis** property group.

**yAxisLabelTextHeight**

Specifies the height in pixels of the y-axis labels.

This property is in the **Y-Axis** property group.

**yAxisMajorDivisions**

Specifies the number of major divisions on the y-axis.

This property is in the **Y-Axis** property group.

**yAxisMinLabelWidth**

Specifies the minimum width in pixels for the y-axis labels.

This property is in the **Y-Axis** property group.

**yAxisMinorDivisions**

Specifies the number of minor divisions on the y-axis.

This property is in the **Y-Axis** property group.

**yAxisMultiRangeFlag**

Select to enable one axis per trace with each trace having its own range.

This property is in the **Y-Axis** property group.

# 12 Table Objects

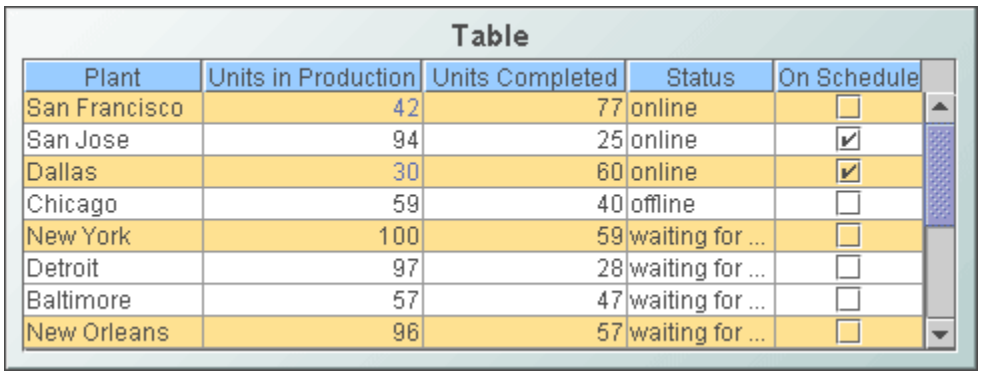

- Standard tables ..... 412
- Rotated tables ..... 435
- HTML5 tables ..... 447

This chapter describes the visualization objects in the **Tables** tab of the Dashboard Builder tool.

## Standard tables

Standard tables display tabular data in a straightforward manner. For each row of the data, there is a row of the displayed table; for each column in the data, there is a column in the displayed table, with the exception of those specified as hidden. Hidden columns can still be used in labels and drill-down substitutions.

Tables are particularly useful as a starting point for drill down. By default, tables are configured to set a number of predefined substitution variables when the end user activates drill down.




Use the valueTable property to attach data to a standard table. Use the columnsToHide property to specify columns to be omitted from the display.

Include a new line character (**\n**) in cell text to display multi-line text.

To copy displayed table data to the system clipboard so that it can be pasted into another application, right click and select **Copy Table Values** or **Copy Cell Value**.

This section covers the following table visualizations:

- Table
- Table with Row Labels
- Table without Grid

These visualizations all share the same properties. They differ from one another only with regard to their default values for these properties. When one of these objects is selected in the Builder canvas, the **Object Class Name** that appears at the top of the **Object Properties** pane is **obj_table02**.

See also "Rotated tables " on page 435, which covers a table with a different set of properties.

The **Object Properties** panel organizes standard table properties into the groups below.

The rule's action is performed on those cells specified by the first field that bear the specified comparison relation to the specified value, or else for the rows that contain

those cells, or for the columns that contain those cells (depending on how the **Target** field is set; see below).

The third field is populated with values from the table's data attachment, based on the selected comparison field, along with the options **top(5)** and **bottom(5)**. Select **top(5)** to specify the five highest values among the cells specified by the first **Condition** field. Select **bottom(5)** to specify the five lowest values among the cells specified by the first **Condition** field. Once you make a selection, you can edit the number in parentheses.

- **Action**: Use this field to specify the rule's action.

In the first drop down menu, select one of the following:

- **Set Background Color To**: controls the color of cell backgrounds.
- **Set Font Color To**: controls the color of cell text.
- **Hide Rows**: controls the visibility of the rows containing the cells specified by the **Condition** fields.
- **Display Image**: replaces cell values with an image. Select the name of the image file from the drop down menu, or enter the pathname of the file (a `.gif`, `.jpg`, or `.png` file). The drop down menu contains the names of image files located in the current directory (typically, the `dashboards` directory of your project directory, under your Apama installation's work directory), as well as image files located in the first level of subdirectories. If you enter a pathname, use an absolute pathname or a pathname that is relative to the current directory.

In the second drop down menu, choose which color to apply to the background or font, enter or select an image, or enter the name of a dashboard variable to use as the value of this field.

- **Target**: Use this field to control the cells to which the action is applied. Select one of the following from the drop down list:
  - **Rows**: applies the action to the rows that contain the cells specified in the **Condition** fields.
  - **Cells**: applies the action to the cells specified in the **Condition** fields.
  - **Columns**: applies the action to the columns that contain the cells specified in the **Condition** fields.

You must select **Rows** if the action is **Hide Rows**.

In the Filter Properties dialog, double-click on an existing rule to edit it. Click the **Remove** button to delete a rule. Click **Clear** to remove all rules. Use the **Move Up** and **Move Down** buttons to control the order in which the rules are applied.

This property is in the **Alert** property group.

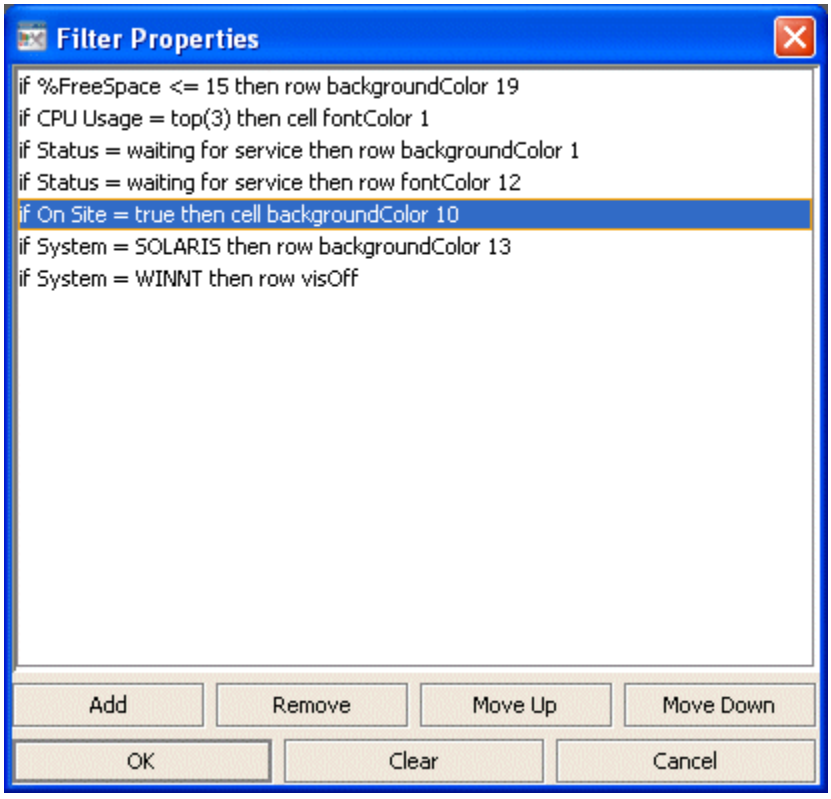
## Standard table: Alert group

This property group contains filterProperties , which specifies how to modify table appearance at runtime in response to changes in the values of individual cells.

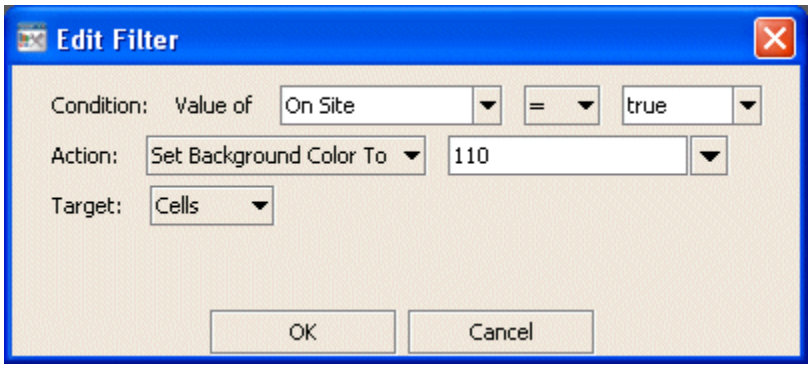
### filterProperties

Specifies rules that are applied at runtime in order to set text color, background color, row visibility, and cell images based on the values of individual cells. The rules are applied in the order in which they are listed in the Filter Properties dialog, therefore later rules may override the effects of earlier rules.

In the Object Properties window, double-click on `filterProperties` in the **Property Name** field. The Filter Properties dialog appears.




Click the **Add** button in the Filter Properties dialog to add a rule. The Edit Filter dialog appears.




The dialog has the following fields and buttons:

- **Condition**: Use these fields to specify the cells for which the rule's action is to be performed. The first field specifies a group of cells and the second and third fields specify a condition. The action is performed for those cells in the specified group that meet the specified condition, or else for the rows containing those cells, or for the columns contain those cells (depending on how the **Target** field is set; see below).

In the first field, supply a column name to specify all cells in a column. Supply **Column Header** to specify all column header cells. Supply **Row Name** for all implicit row name cells.

In the second field, select a comparison relation.

In the third field, select or enter a value or dashboard variable name.

## Standard table: Background group

Properties in this group control the visibility and appearance of the rectangle that serves as the background of the table and the table's main label (see label).

### Background group properties

The group contains the following properties:


- "bgBorderColor" on page 415
- "bgBorderFlag" on page 416
- "bgColor" on page 416
- "bgEdgeWidth" on page 416
- "bgGradientColor2" on page 416
- "bgGradientMode" on page 416
- "bgRaisedFlag" on page 417
- "bgRoundness" on page 417
- "bgShadowFlag" on page 417
- "bgStyleFlag" on page 417
- "bgVisFlag" on page 417
- "borderPixels" on page 417
- "tableBgColor" on page 418


### bgBorderColor

Sets the color of the border (see `bgBorderFlag`) of the background rectangle. Select the **...** button and choose a color from the palette. Close the Color Chooser window when you are done.

#### bgBorderFlag

Select to display a border around the background rectangle.

This property is in the **Background** property group.

#### bgColor

Sets the background color. Select the **...** button and choose a color from the palette. Close the Color Chooser window when you are done.

This property is in the **Background** property group.

#### bgEdgeWidth

Sets the width in pixels of the 3D edge on the background rectangle. This property is only used if bgBorderFlag is selected.

This property is in the **Background** property group.

#### bgGradientColor2

Sets the color for the second color in the gradient. The default is white. The `bgColor` property sets the first color in the gradient.

This property is in the **Background** property group.

#### bgGradientMode

Display a gradient in the background rectangle. Select from the following options:

- **None**: No gradient
- **Diagonal Edge**: Gradient is drawn at a 45 degree angle from the top left to the bottom right corner of the object.
- **Diagonal Center**: Gradient is drawn at a 45 degree angle from the center to the top left and the bottom right corners of the object.
- **Horizontal Edge**: Gradient is drawn horizontally from the top to the bottom of the object.
- **Horizontal Center**: Gradient is drawn horizontally from the center to the top and bottom of the object.
- **Vertical Edge**: Gradient is drawn vertically from the left to the right of the object.
- **Vertical Center**: Gradient is drawn vertically from the center to the left and right of the object.

This property is in the **Background** property group.

#### bgRaisedFlag

Reverses the direction of the gradient, as well as that of the 3D edge if the `bgStyle` selected is 3D Rectangle.

This property is in the **Background** property group.

#### bgRoundness

Sets the arc length of the rounded corners. This property is only available if the `bgStyle` selected is **Round Rectangle**.

The value of `bgRoundness` cannot exceed half the value of the `objWidth` or the `objHeight`. If `bgRoundness` does exceed that value, half of `objWidth` or `objHeight` (whichever is smaller) will be used instead. For example if `objWidth` is `100` and `objHeight` is `50`, then the value of `bgRoundness` cannot exceed `25`. If it does, then half the value of `objHeight` (`25`) will be used instead. This property is in the **Background** property group.

#### bgShadowFlag

Select to display a drop shadow on the background rectangle.

This property is in the **Background** property group.

#### bgStyleFlag

Choose one of the following three options from the drop down menu:

- **Rectangle**: Select to display a background rectangle.
- **3D Rectangle**: Select to display a 3D edge on the background rectangle. If selected, use `bgEdgeWidth` to set the width of the 3D edge.
- **Round Rectangle**: Select to display a background rectangle with rounded edges. If selected, use `bgRoundness` to set the arc length of the rounded corners.

This property is in the **Background** property group.

#### bgVisFlag

Select to display the background rectangle.

This property is in the **Background** property group.

#### borderPixels

Sets the width in pixels of the border between the chart and the edge of the background rectangle.

This property is in the **Background** property group.

#### tableBgColor

Sets the color of empty space in the table. Select the **...** button and choose a color from the palette. Close the Color Chooser window when you are done.

This property is in the **Background** property group.

## Standard table: Cell group

Properties in this group control the appearance of cell text and cell background color.

#### Cell group properties

The group contains the following properties:

- "cellBgColor" on page 418
- "cellBgStripedContrast" on page 418
- "cellBgStripedFlag" on page 418
- "cellTextColor" on page 418
- "cellTextFont" on page 419
- "cellTextSize" on page 419

#### cellBgColor

Sets the background color of the cells. Select the **...** button and choose a color from the palette. Close the Color Chooser window when you are done.

This property is in the **Cell** property group.

#### cellBgStripedContrast

Specifies the contrast level for the stripes if cellBgStripedFlag is enabled.

This property is in the **Cell** property group.

#### cellBgStripedFlag

Specifies alternating striped rows. Alternate rows have a lighter shade of the color specified in cellBgColor.

This property is in the **Cell** property group.

#### cellTextColor

Sets the text color of the cells. Select the **...** button and choose a color from the palette. Close the Color Chooser window when you are done.

This property is in the **Cell** property group.

#### cellTextFont

Sets the font for cell text. Select an item from the drop down menu.

This property is in the **Cell** property group.

#### cellTextSize

Sets the point size of the cell text. The default is 11 points. If you enter a negative value, the default is used.

This property is in the **Cell** property group.

## Standard table: Column group

Properties in this group control the visibility, width, and resize behavior of table columns, as well as the format and alignment of cell text within each column.

#### Column group properties

The group contains the following properties:

- "autoResizeFlag" on page 419
- "columnAlignment" on page 419
- "columnFormat" on page 420
- "columnProperties" on page 421
- "columnsToHide" on page 422
- "indexColumns" on page 423

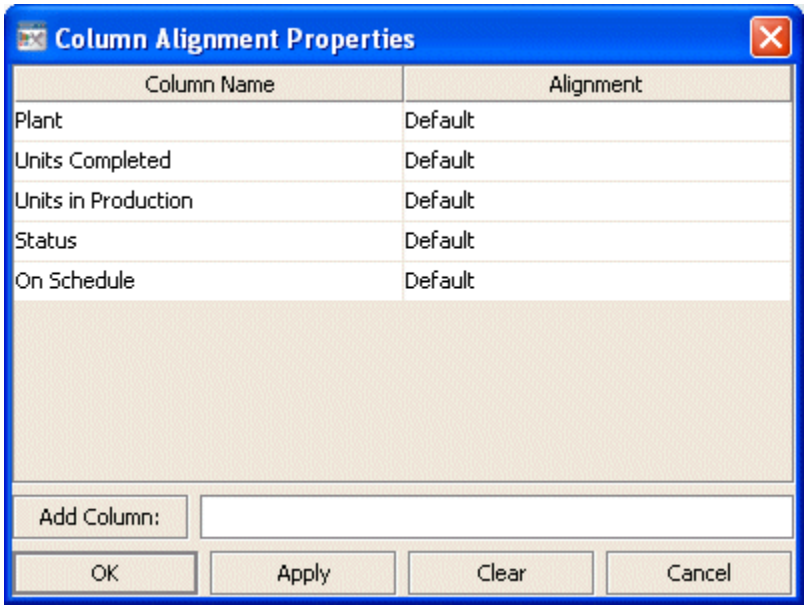
#### autoResizeFlag

When this property is enabled, column widths change automatically to accommodate table resizing. When this property is disabled, column width is fixed at the values specified by columnProperties.

This property is in the **Column** property group.

#### columnAlignment

Specifies the alignment of text within each column. Select the ellipsis button. The Column Alignment Properties dialog appears.




Select one of the following alignment specifications from the drop down menu in the Alignment column:

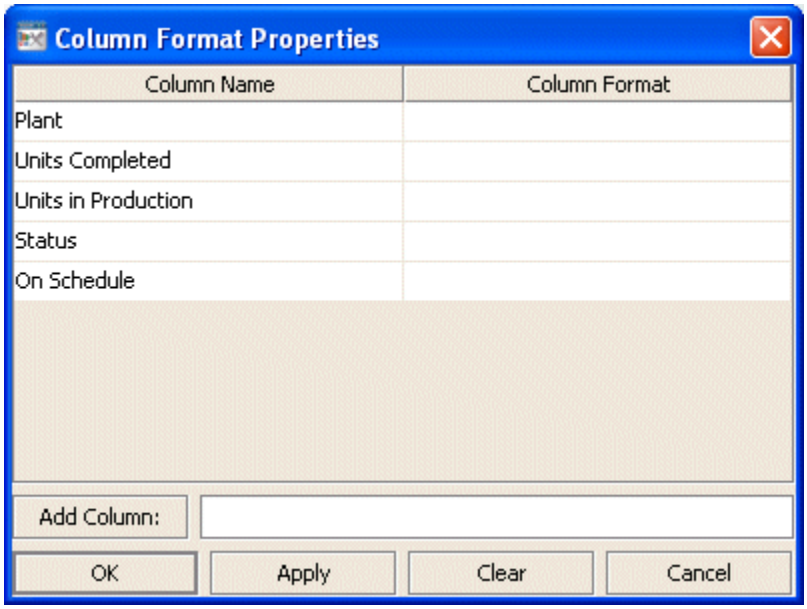
- **Default**
- **Left**
- **Center**
- **Right**

The default setting depends on the type of column.

This property is in the **Column** property group.

#### columnFormat

Specifies formats for numerical and date columns. In the Object Properties window, double-click on `columnFormat` in the **Property Name** field. The Column Format Properties dialog appears.




The **Column Name** list is populated based on the table's data attachment. If you have not yet attached the table to data, this list is empty.

In the **Column Format** column of the dialog, enter a format or select a format from the drop down menu, and press Enter. Specify numerical formats based on the Java format specification, or with the following shorthand:

- **$** for US dollar money values
- **$$** for US dollar money values with additional formatting, () for non-money values, formatted similar to money
- **#** for positive or negative whole values

Specify date formats based on the Java date specification.

The dialog has the following buttons:

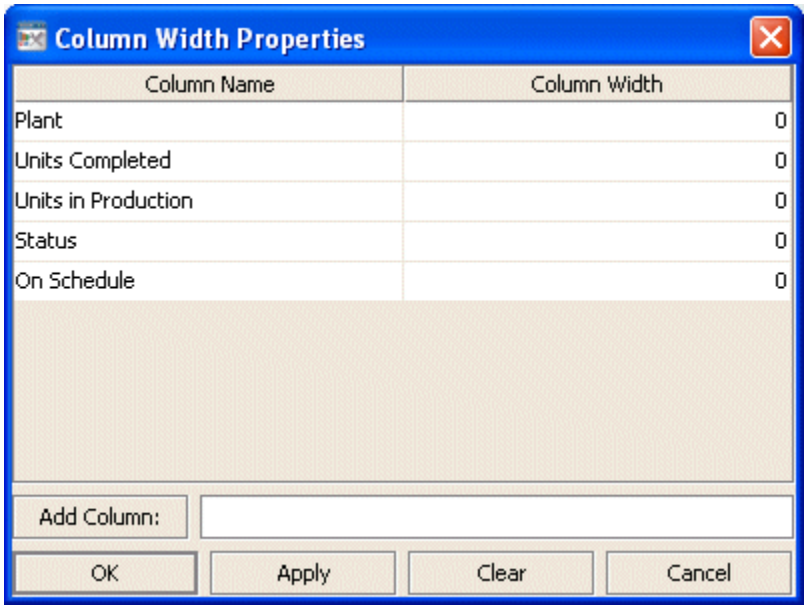
- **Add Column:** Enter the name of the column and click the **Add Column** button to insert a column into the table.
- **Clear:** Click the **Clear** button to clear all cells in the **Column Format** column of the dialog.

> **Note:** Text columns that contain text representing numbers are treated as if they are numeric columns, so number formats can be applied.

This property is in the **Column** property group.

#### columnProperties

Specifies the width of each column. In the Object Properties window, double-click on `columnProperties` in the **Property Name** field. The Column Width Properties dialog appears.




The **Column Name** list is populated based on the table's data attachment. If you have not yet attached the table to data, this list is empty.

In the **Column Width** column of the dialog, enter the width in pixels.

The dialog has the following buttons:

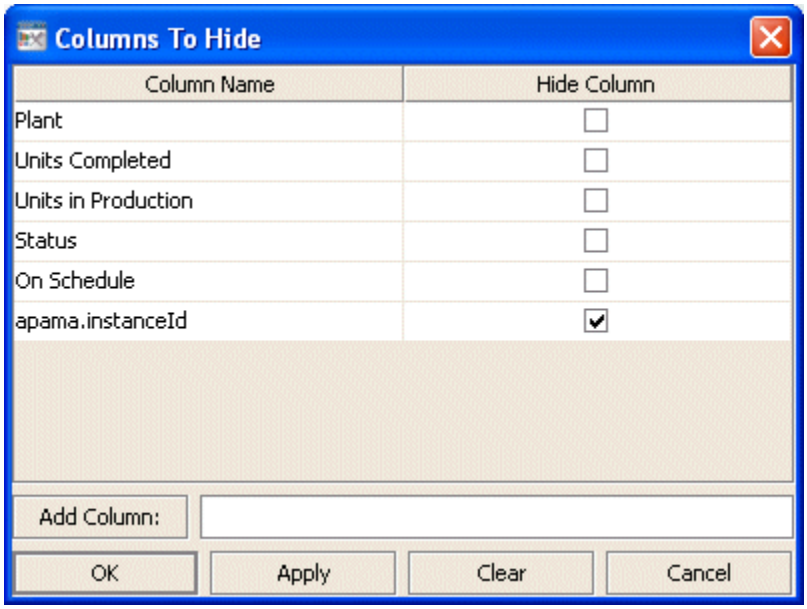
- **Add Column:** Enter the name of the column and click the **Add Column** button to insert a column into the table.
- **Clear:** Click the **Clear** button to clear all cells in the **Column Width** column of the dialog.

This property is in the **Column** property group.

#### columnsToHide

Specifies columns from the data attachment to exclude from being displayed in the table. Columns specified in the `columnsToHide` property can still be used in the drillDownColumnSubs property.

In the Object Properties window, double-click on `columnsToHide` in the **Property Name** field. The Columns To Hide dialog appears.




The **Column Name** list is populated based on the table's data attachment. If you have not yet attached the table to data, this list is empty.

In the **Hide Column** column of the dialog, click the checkbox for each column that you want to hide.

The dialog has the following buttons:

- **Add Column:** Enter the name of the column and click the **Add Column** button to insert a column into the table.
- **Clear:** Click the **Clear** button to clear all cells in the **Column Width** column of the dialog.

Dashboard Builder displays a warning message if you attempt to hide the row header column.

This property is in the **Column** property group.

#### indexColumns

Use this property in order to maintain the highlight of selected rows after data updates or table sorts are executed. In the **Object Properties** window, double-click on `indexColumns` in the **Property Name** field. The Index Columns dialog appears. Select one or more columns whose values uniquely identify each row.

This property is in the **Column** property group.

## Standard table: Column Header group

Properties in this group control the color, font, and size of column-header text, as well as the column-header background color.

#### Column Header group properties

The group contains the following properties:

- "columnHeaderBgColor" on page 424
- "columnHeaderTextColor" on page 424
- "columnHeaderTextFont" on page 424
- "columnHeaderTextSize" on page 424

#### columnHeaderBgColor

Sets the background color of the column headers. Select the **...** button and choose a color from the palette. Close the Color Chooser window when you are done.

This property is in the **Column Header** property group.

#### columnHeaderTextColor

Sets the text color of the column headers. Select the **...** button and choose a color from the palette. Close the Color Chooser window when you are done.

This property is in the **Column Header** property group.

#### columnHeaderTextFont

Sets the font for column header text. Select an item from the drop down menu.

This property is in the **Column Header** property group.

#### columnHeaderTextSize

Sets the point size of the column header text. The default is 11 points. If you enter a negative value, the default is used.

This property is in the **Column Header** property group.

## Standard table: Data group

Properties in this group determine what data populates the table.

#### Data group properties

The group contains the following properties:

- "insertNewRowsAtTopFlag" on page 425
- "insertNewRowsFlag" on page 425
- "maxNumberOfRows" on page 425
- "rowLabelMode" on page 425
- "valueTable" on page 425

#### insertNewRowsAtTopFlag

Controls whether new rows are inserted at the top or bottom of the table, if insertNewRowsFlag is enabled.

This property is in the **Data** property group.

#### insertNewRowsFlag

Controls whether the table contents are replaced or augmented with new data sent to the dashboard. If this property is enabled, the table contents are augmented through the addition of new rows. If this property is not enabled, new data replaces the table contents.

This property is in the **Data** property group.

#### maxNumberOfRows

Sets the maximum number of rows that the table can contain. Enter a value that is less than or equal to 131072.

This property is in the **Data** property group.

#### rowLabelMode

Supply a positive value to enable a row-header column consisting of generated row IDs.

This property is in the **Data** property group.

#### valueTable

Attach your data to the `valueTable` property. Right-click on the property name in the Object Properties panel, and select a menu item under **Attach to Data**.

This property is in the **Data** property group.

## Standard table: Data Label group

This property group contains the property columnDisplayNames, which specifies non-default column-header text.

#### columnDisplayNames

Sets alternate display names for the columns of the data attached to valueTable.

This property is in the **Data Label** property group.

## Standard table: Grid group

Properties in this group control the visibility of the horizontal and vertical lines that separate table rows and columns.

#### Grid group properties

The group contains the following properties:

- "gridHorizontalVisFlag" on page 426
- "gridVerticalVisFlag" on page 426

#### gridHorizontalVisFlag

Controls the visibility of the horizontal lines that separate table rows.

This property is in the **Grid** property group.

#### gridVerticalVisFlag

Controls the visibility of the vertical lines that separate table columns.

This property is in the **Grid** property group.

## Standard table: Historian group

Do not use the properties in this group.

#### historyTableName

Do not use this property.

This property is in the **Historian** property group.

#### historyTableRowNameFlag

Do not use this property.

This property is in the **Historian** property group.

## Standard table: Interaction group

Properties in this group control various forms of interaction between the end user and the table, including scrolling, highlighting, selecting rows, and activating commands, drill downs, and tooltips. There is also a property that controls end-user keyboard navigation with the Tab key. See also "Standard table: Sort group" on page 434.

#### Interaction group properties

The group contains the following properties:

- "columnResizeEnabledFlag" on page 427
- "command" on page 427
- "commandCloseWindowOnSuccess" on page 428
- "commandConfirm" on page 428

- "confirmText" on page 428
- "drillDownColumnSubs" on page 428
- "drillDownSelectMode" on page 429
- "drillDownTarget" on page 429
- "editDataEnabledFlag" on page 429
- "editDataLocalVarName" on page 429
- "multiSelectFlag" on page 429
- "rowHighlightEnabledFlag" on page 430
- "scrollToSelectionFlag" on page 430
- "scrollbarMode" on page 430
- "tabIndex" on page 430


#### columnResizeEnabledFlag

If selected, the end user can resize table columns by dragging the vertical separators between the column headers. This property also enables resize by dragging for the Dashboard Builder.

> **Note:** This property is ignored for thin client (Display Server) deployments.

This property is in the **Interaction** property group.

#### command

Assign a command or group of commands to this stock chart by right-clicking on the `command` property name in the Object Properties window. Select **Define Command** and choose **SYSTEM**, **APAMA**, or **MULTIPLE**. See "Using the Define Apama Command dialog" on page 226.

Once a command or command group has been assigned to this object, you can activate it from a deployed dashboard or from the Dashboard Builder:

- Dashboard Builder: Double click on the object.
- Web-based deployment: Single click on the object or else right click on it and select **Execute Command** from the popup menu.
- Local deployment: By default, single-click on the object or else right-click on it and select **Execute Command** from the popup menu. To override the default, select **Tools > Options** in the Builder (do this before you generate the deployment package), and uncheck **Single-Click for Drill Down and Commands** in the **General** tab. This allows the end user to use either a double click or a right click.

When you activate a command, any defined drill down substitutions are performed, and then the command is executed.

If you assign multiple commands, the commands are launched in an arbitrary order, and are executed asynchronously; there is no guarantee that one command will finish before the next one in the sequence starts.

This property is in the **Interaction** property group.

#### commandCloseWindowOnSuccess

Select this property to automatically close the window that initiates a **SYSTEM** command when the command is executed successfully. This applies to **SYSTEM** commands only, and is not supported at all for thin-client (Display Sever) deployments.

With **APAMA** commands, the window is closed whether or not the command is executed successfully. For **MULTIPLE** commands, the window closes when the first command in the command group succeeds.

This property is in the **Interaction** property group.

#### commandConfirm

By default, when the end user executes a command (see the command property), the command confirmation dialog is disabled. To control this option for each individual object, use the `commandConfirm` check box. If confirmation is required for a **MULTIPLE** command group, a single confirmation dialog is presented; if you confirm the execution, all individual commands in the group are executed with no further confirmation. If the you cancel the execution, none of the commands in the group is executed.

You can also override the confirmation status of individual objects with an application-wide policy. Select **Tools > Options** and choose from three confirmation values:

- **Do not confirm**: Indicates that no commands require confirmation (regardless of each object's confirmation status).
- **Confirm all**: Indicates that all commands require confirmation (regardless of each object's confirmation status).
- **Use object confirm flag** (default): Indicates that the confirmation status of each object will determine whether confirmation is required.

This property is in the **Interaction** property group.

#### confirmText

Use this property to write your own text for the confirmation dialog. Otherwise, default text is used. See commandConfirm.

This property is in the **Interaction** property group.

#### drillDownColumnSubs

Use this property to direct a dashboard to assign data-table column values to specified dashboard variables when the end user activates a drilldown on this object. In the Object Properties window, double-click on `drillDownColumnSubs` in the **Property Name** field to bring up the Drill Down Column Substitutions dialog.

The dialog has the following fields and buttons:

- **Substitution String**: Enter the dashboard variable next to the name of the data table column whose value you want assigned to the variable. Press Enter.
- **Add Column**: Enter the name of a column and click the **Add Column** button to insert a column into the table.
- **Clear**: Click the **Clear** button to remove all variables listed.

The **Column Name** list is populated based on the table's data attachment. If you have not yet attached the table to data, this list is empty.

Once you have selected which column values to pass in as substitutions, double-click on any element in your object to open a drill down window that displays corresponding values.

This property is in the **Interaction** property group.

#### drillDownSelectMode

Use this property to control how a drill down display is activated. Select one of the following:

- **Anywhere** to activate a drill down display by double-clicking anywhere on the chart.
- **Element Only** to enable a drill down display only when you double-click on an element of the chart, such as a bar or candlestick.

This property is in the **Interaction** property group.

#### drillDownTarget

To specify a drill down display, double click on `drillDownTarget` in the **Property Name** field to bring up the Drill Down Properties dialog. See "Drill-Down Specification" on page 535.

This property is in the **Interaction** property group.

#### editDataEnabledFlag

Do not use this property.

This property is in the **Interaction** property group.

#### editDataLocalVarName

Do not use this property.

This property is in the **Interaction** property group.

#### multiSelectFlag

Enables the selection of multiple rows. When the user selects multiple rows and drills down, the drill down substitution values contain a semi colon delimited list of values,

one value for each row that can be used with most data sources in the **Filter** fields of the Attach To Data dialogs.

This property is in the **Interaction** property group.

#### rowHighlightEnabledFlag

Enables highlighting of an entire row when a cell in the row is selected by the end user.

This property is in the **Interaction** property group.

#### scrollToSelectionFlag

When this property is enabled, the selected row is made visible whenever the table is updated or redrawn. If multiple rows are selected, the topmost selected row is made visible.

This property is in the **Interaction** property group.

#### scrollbarMode

Select one of the following to set the behavior of the table scroll bars:

- **Never**: Default setting. Some rows or columns may get clipped.
- **As Needed**: Display a scroll bar or scroll bars when there is not enough space to display all of the rows or columns.
- **Always**: Display scroll bars at all times.

This property is in the **Interaction** property group.

#### tabIndex

Defines the order in which this table object receives focus (relative to other table objects and control objects) during keyboard navigation using the Tab key. Initial focus is given to the object with the smallest positive `tabIndex` value. The tabbing order proceeds in ascending order. If multiple objects share the same `tabIndex` value, initial focus and tabbing order are determined by the alpha-numeric order of the table names. Tables with a `tabIndex` value of 0 are last in the tabbing order.

**Note:** This property does not apply to thin-client (Display Server) deployments, or to objects that are disabled, invisible, or have a value of less than 0.

This property is in the **Interaction** property group.

## Standard table: Label group

Properties in this group control the table's main label, including text, alignment, color, font, and size.

#### Label group properties

The group contains the following properties:

- "label" on page 431
- "labelTextAlignX" on page 431
- "labelTextColor" on page 431
- "labelTextFont" on page 431
- "labelTextHeight" on page 431


#### label

Specifies the text for the chart label. Click the ellipsis for multi-line text.

The default is **Table**, **Table with Row Labels**, or **Table without Grid**.

This property is in the **Label** property group.

#### labelTextAlignX

Sets the alignment of the chart label (see the label property). Select **Left**, **Center**, or **Right** from the drop down list.

This property is in the **Label** property group.

#### labelTextColor

Specifies the color of the chart label text (see the label property). Select the **...** button and choose a color from the palette. Close the Color Chooser window when you are done.

This property is in the **Label** property group.

#### labelTextFont

Specifies the font of the chart label text (see the label property). Select an item from drop down list.

This property is in the **Label** property group.

#### labelTextHeight

Specifies the point size of the chart label text (see the label property).

This property is in the **Label** property group.

## Standard table: Object group

Properties in this group control the visibility and transparency of the table as a whole. They also control (or reflect) the overall position and dimensions of the table. In addition, a property in this group reflects the generated name of this individual table.

#### Object group properties

The group contains the following properties:


- "anchor" on page 432

- "dock" on page 432
- "objHeight" on page 432
- "objName" on page 432
- "objWidth" on page 432
- "objX" on page 433
- "objY" on page 433
- "visFlag" on page 433


#### anchor

Select zero or more of **Top**, **Left**, **Bottom**, and **Right** in order to control the object's placement. The `anchor` property is only applied when the display is resized either by changing the **Background Properties** on the display or by resizing the window in **Layout** mode. If an object has the `dock` property set, the `anchor` property is ignored. See "About resize modes" on page 39.

#### dock

Select **None** (default), **Top**, **Left**, **Bottom**, **Right**, or **Fill** in order to control the object's placement in **Layout** resize mode. See "About resize modes" on page 39.

#### objHeight

Set the height of a chart by entering a value for this property or by dragging a handle of the bounding box that appears when the chart is selected. When you drag a handle of the bounding box, the displayed value for this property changes to reflect the real-time height of the chart.

This property is in the **Object** property group.

#### objName

An identifier that is generated by the Dashboard Builder. This name can be used by other objects' properties in order to refer to the named chart.

This property is in the **Object** property group.

#### objWidth

Set the width of a chart by entering a value for this property or by dragging a handle of the bounding box that appears when the chart is selected. When you drag a handle of the bounding box, the displayed value for this property changes to reflect the real-time width of the chart.

This property is in the **Object** property group.

#### objX

Sets the X coordinate of the center of this visualization object, relative to the lower left corner of the current dashboard. This value is set automatically when you position the object with the mouse.

This property is in the **Object** property group.

#### objY

Sets the Y coordinate of the center of this visualization object, relative to the lower left corner of the current dashboard. This value is set automatically when you position the object with the mouse.

This property is in the **Object** property group.

#### visFlag

Deselect to make this visualization object invisible in the current dashboard.

This property is in the **Object** property group.

## Standard table: Row Header group

A property in this group specifies whether the cells in the first column are considered to be row headers. If this property is selected, other properties control row-header text color, font, point size, and background color.

#### Row Header group properties

The group contains the following properties:

- "rowHeaderBgColor" on page 433
- "rowHeaderEnabledFlag" on page 434
- "rowHeaderFilterColorsEnabledFlag" on page 434
- "rowHeaderTextColor" on page 434
- "rowHeaderTextFont" on page 434
- "rowHeaderTextSize" on page 434

#### rowHeaderBgColor

Sets the background color of row-header cells, provided rowHeaderEnabledFlag is enabled. Select the **...** button and choose a color from the palette. Close the Color Chooser window when you are done.

This property is in the **Row Header** property group.

#### rowHeaderEnabledFlag

Specifies that the cells in the first column are row headers When this property is enabled, you can set the appearance of the row-header column using rowHeaderBgColor, rowHeaderTextColor, rowHeaderTextFont, and rowHeaderTextSize properties.

Dashboard Builder displays a warning message if you attempt to hide the row-header column by using the columnsToHide property.

This property is in the **Row Header** property group.

#### rowHeaderFilterColorsEnabledFlag

Disable this property to disable the effect of filterProperties on the background color or text color of cells in the row-header column, provided rowHeaderEnabledFlag is enabled. Note that this does not override the effects of filterProperties on row visibility.

This property is in the **Row Header** property group.

#### rowHeaderTextColor

Sets the text color for row-header cells, provided rowHeaderEnabledFlag is enabled. Select the **...** button and choose a color from the palette. Close the Color Chooser window when you are done.

This property is in the **Row Header** property group.

#### rowHeaderTextFont

Sets the font for row-header cells, provided rowHeaderEnabledFlag is enabled. Select an item from the drop down list.

This property is in the **Row Header** property group.

#### rowHeaderTextSize

Sets the text point size for row-header cells, provided rowHeaderEnabledFlag is enabled.

This property is in the **Row Header** property group.

## Standard table: Sort group

Properties in this group control the order in which table rows appear, as well as whether the end user can re-sort table rows by clicking on a column header.

#### Sort group properties

The group contains the following properties:

- "showSortIconFlag" on page 435
- "sortAscendingFlag" on page 435

- "sortColumnName" on page 435

#### showSortIconFlag

When this property is enabled, the end user can click a column's header to sort the table rows according to the values in that column. The sortAscendingFlag property determines whether the sort is initially ascending or descending. Clicking a column header again reverses the sort order. In Dashboard Builder, sortAscendingFlag changes in real time to reflect the current sort order, and sortColumnName changes in real time to reflect the current sort column.

In addition, when this property is enabled, a sort icon (an arrow head) appears next to the header of the current sort column (determined initially by sortColumnName). The direction in which the arrow head points indicates whether the current sort order is ascending or descending.

This property is in the **Sort** property group.

#### sortAscendingFlag

Determines whether the current sort order is ascending or descending. See sortColumnName and showSortIconFlag.

This property is in the **Sort** property group.

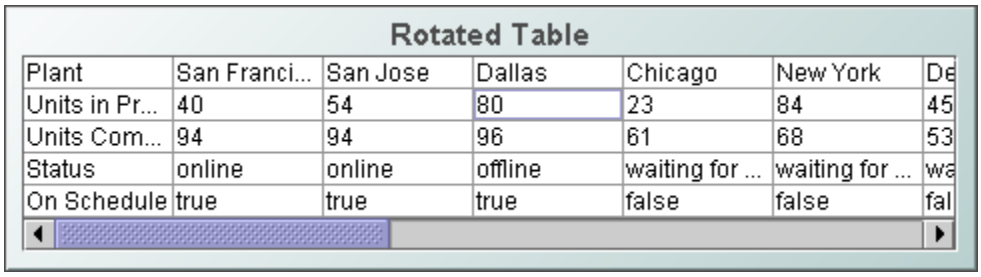
#### sortColumnName

Sets the column whose values determine the order in which table rows appear. If sortAscendingFlag is enabled, rows with earlier values (either numerically or alphabetically) appear first. See also showSortIconFlag.

This property is in the **Sort** property group.

## Rotated tables

Rotated tables display tabular data by swapping rows and columns. For each row of the data, there is a column in the displayed table; for each column in the data, there is a row in the displayed table.




Use the valueTable property to attach data to a rotated table.

Include a new line character (`\n`) in the cell text to display multi-line text.

To copy data to the system clipboard so that it can be pasted into another application, right-click and select **Copy Table Values** or **Copy Cell Value**.

When a rotated table is selected in the Builder canvas, the **Object Class Name** that appears at the top of the **Object Properties** pane is **obj_table03**.

The **Object Properties** panel organizes rotated table properties into the groups below.

## Rotated table: Background group

Properties in this group control the visibility and appearance of the rectangle that serves as the background of the table and the table's main label (see label).

### Background group properties

The group contains the following properties:


- "bgBorderColor" on page 436
- "bgBorderFlag" on page 436
- "bgColor" on page 437
- "bgEdgeWidth" on page 437
- "bgGradientColor2" on page 437
- "bgGradientMode" on page 437
- "bgRaisedFlag" on page 437
- "bgRoundness" on page 438
- "bgShadowFlag" on page 438
- "bgStyleFlag" on page 438
- "bgVisFlag" on page 438
- "borderPixels" on page 438


### bgBorderColor

Sets the color of the border (see `bgBorderFlag`) of the background rectangle. Select the **...** button and choose a color from the palette. Close the **Color Chooser** window when you are done.

### bgBorderFlag

Select to display a border around the background rectangle.

This property is in the **Background** property group.

#### bgColor

Sets the background color. Select the **...** button and choose a color from the palette. Close the Color Chooser window when you are done.

This property is in the **Background** property group.

#### bgEdgeWidth

Sets the width in pixels of the 3D edge on the background rectangle. This property is only used if bgBorderFlag is selected.

This property is in the **Background** property group.

#### bgGradientColor2

Sets the color for the second color in the gradient. The default is white. The `bgColor` property sets the first color in the gradient.

This property is in the **Background** property group.

#### bgGradientMode

Display a gradient in the background rectangle. Select from the following options:

- **None**: No gradient
- **Diagonal Edge**: Gradient is drawn at a 45 degree angle from the top left to the bottom right corner of the object.
- **Diagonal Center**: Gradient is drawn at a 45 degree angle from the center to the top left and the bottom right corners of the object.
- **Horizontal Edge**: Gradient is drawn horizontally from the top to the bottom of the object.
- **Horizontal Center**: Gradient is drawn horizontally from the center to the top and bottom of the object.
- **Vertical Edge**: Gradient is drawn vertically from the left to the right of the object.
- **Vertical Center**: Gradient is drawn vertically from the center to the left and right of the object.

This property is in the **Background** property group.

#### bgRaisedFlag

Reverses the direction of the gradient, as well as that of the 3D edge if the `bgStyle` selected is 3D Rectangle.

This property is in the **Background** property group.

#### bgRoundness

Sets the arc length of the rounded corners. This property is only available if the `bgStyle` selected is **Round Rectangle**.

The value of `bgRoundness` cannot exceed half the value of the `objWidth` or the `objHeight`. If `bgRoundness` does exceed that value, half of `objWidth` or `objHeight` (whichever is smaller) will be used instead. For example if `objWidth` is `100` and `objHeight` is `50`, then the value of `bgRoundness` cannot exceed `25`. If it does, then half the value of `objHeight` (`25`) will be used instead. This property is in the **Background** property group.

#### bgShadowFlag

Select to display a drop shadow on the background rectangle.

This property is in the **Background** property group.

#### bgStyleFlag

Choose one of the following three options from the drop down menu:

- **Rectangle**: Select to display a background rectangle.
- **3D Rectangle**: Select to display a 3D edge on the background rectangle. If selected, use `bgEdgeWidth` to set the width of the 3D edge.
- **Round Rectangle**: Select to display a background rectangle with rounded edges. If selected, use `bgRoundness` to set the arc length of the rounded corners.

This property is in the **Background** property group.

#### bgVisFlag

Select to display the background rectangle.

This property is in the **Background** property group.

#### borderPixels

Sets the width in pixels of the border between the chart and the edge of the background rectangle.

This property is in the **Background** property group.

## Rotated table: Cell group

Properties in this group control the appearance of cell text and cell background color.

#### Cell group properties

The group contains the following properties:

- "cellBgColor" on page 439

- "cellBgStripedContrast" on page 439
- "cellBgStripedFlag" on page 439
- "cellTextColor" on page 439
- "cellTextFont" on page 439
- "cellTextSize" on page 439


#### cellBgColor

Sets the background color of the cells. Select the **...** button and choose a color from the palette. Close the Color Chooser window when you are done.

This property is in the **Cell** property group.

#### cellBgStripedContrast

Specifies the contrast level for the stripes if cellBgStripedFlag is enabled.

This property is in the **Cell** property group.

#### cellBgStripedFlag

Specifies alternating striped rows. Alternate rows have a lighter shade of the color specified in cellBgColor.

This property is in the **Cell** property group.

#### cellTextColor

Sets the text color of the cells. Select the **...** button and choose a color from the palette. Close the Color Chooser window when you are done.

This property is in the **Cell** property group.

#### cellTextFont

Sets the font for cell text. Select an item from the drop down menu.

This property is in the **Cell** property group.

#### cellTextSize

Sets the point size of the cell text.

This property is in the **Cell** property group.

## Rotated table: Column group

Properties in this group control the width and resize-behavior of table columns.

#### Column group properties

The group contains the following properties:

- "autoResizeFlag" on page 440
- "columnProperties" on page 440

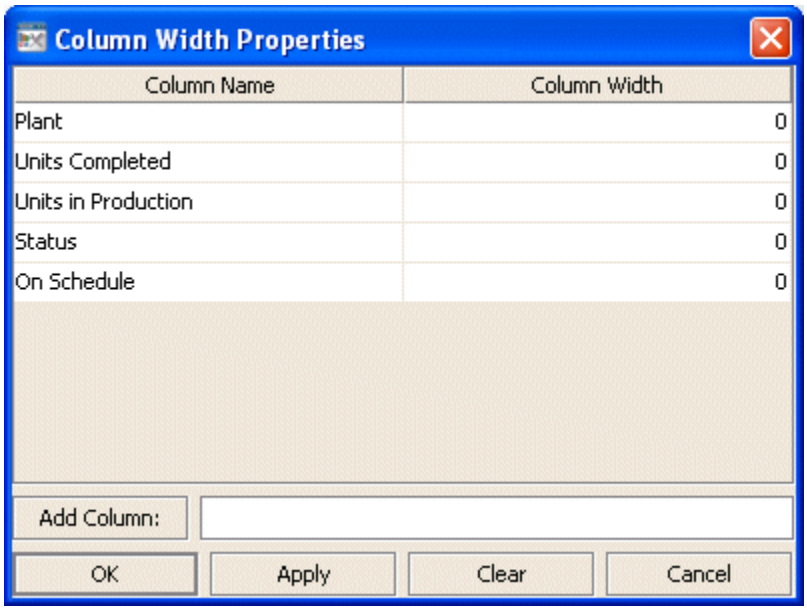

#### autoResizeFlag

When this property is enabled, column widths change automatically to accommodate table resizing. When this property is disabled, column width is fixed at the values specified by columnProperties.

This property is in the **Column** property group.

#### columnProperties

Specifies the width of each column. In the Object Properties window, double-click on `columnProperties` in the **Property Name** field. The Column Width Properties dialog appears.




The **Column Name** list is populated based on the table's data attachment. If you have not yet attached the table to data, this list is empty.

In the **Column Width** column of the dialog, enter the width in pixels.

The dialog has the following buttons:

- **Add Column:** Enter the name of the column and click the **Add Column** button to insert a column into the table.
- **Clear:** Click the **Clear** button to clear all cells in the **Column Width** column of the dialog.

This property is in the **Column** property group.

## Rotated table: Data group

The property in this group, valueTable, determines what data populates the table.

#### valueTable

Attach your data to the `valueTable` property. Right-click on the property name in the Object Properties panel, and select a menu item under **Attach to Data**.

This property is in the **Data** property group.

## Rotated table: Grid group

Properties in this group control the visibility of the horizontal and vertical lines that separate table rows and columns.

#### Grid group properties

The group contains the following properties:

- "gridHorizontalVisFlag" on page 441
- "gridVerticalVisFlag" on page 441

#### gridHorizontalVisFlag

Controls the visibility of the horizontal line that separate table rows.

This property is in the **Grid** property group.

#### gridVerticalVisFlag

Controls the visibility of the vertical line that separate table columns.

This property is in the **Grid** property group.

## Rotated table: Historian group

Do not use the properties in this group.

#### historyTableName

Do not use this property.

This property is in the **Historian** property group.

#### historyTableRowNameFlag

Do not use this property.

This property is in the **Historian** property group.

## Rotated table: Interaction group

Properties in this group control command and drill-down of interaction between the end user and the table.

### Interaction group properties

The group contains the following properties:


- "command" on page 442
- "commandCloseWindowOnSuccess" on page 443
- "commandConfirm" on page 443
- "confirmText" on page 443
- "drillDownColumnSubs" on page 443
- "drillDownSelectMode" on page 444
- "drillDownTarget" on page 444


### command

Assign a command or group of commands to this stock chart by right-clicking on the `command` property name in the Object Properties window. Select **Define Command** and choose **SYSTEM**, **APAMA**, or **MULTIPLE**. See "Using the Define Apama Command dialog" on page 226.

Once a command or command group has been assigned to this object, you can activate it from a deployed dashboard or from the Dashboard Builder:

- Dashboard Builder: Double click on the object.
- Web-based deployment: Single click on the object or else right click on it and select **Execute Command** from the popup menu.
- Local deployment: By default, single-click on the object or else right-click on it and select **Execute Command** from the popup menu. To override the default, select **Tools > Options** in the Builder (do this before you generate the deployment package), and uncheck **Single-Click for Drill Down and Commands** in the **General** tab. This allows the end user to use either a double click or a right click.

When you activate a command, any defined drill down substitutions are performed, and then the command is executed.

If you assign multiple commands, the commands are launched in an arbitrary order, and are executed asynchronously; there is no guarantee that one command will finish before the next one in the sequence starts.

This property is in the **Interaction** property group.

#### commandCloseWindowOnSuccess

Select this property to automatically close the window that initiates a **SYSTEM** command when the command is executed successfully. This applies to **SYSTEM** commands only, and is not supported at all for thin-client, Web-page deployments.

With **APAMA** commands, the window is closed whether or not the command is executed successfully. For **MULTIPLE** commands, the window closes when the first command in the command group succeeds.

This property is in the **Interaction** property group.

#### commandConfirm

By default, when the end user executes a command (see the command property), the command confirmation dialog is disabled. To control this option for each individual object, use the `commandConfirm` check box. If confirmation is required for a **MULTIPLE** command group, a single confirmation dialog is presented; if you confirm the execution, all individual commands in the group are executed with no further confirmation. If the you cancel the execution, none of the commands in the group is executed.

You can also override the confirmation status of individual objects with an application-wide policy. Select **Tools > Options** and choose from three confirmation values:

- **Do not confirm**: Indicates that no commands require confirmation (regardless of each object's confirmation status).
- **Confirm all**: Indicates that all commands require confirmation (regardless of each object's confirmation status).
- **Use object confirm flag** (default): Indicates that the confirmation status of each object will determine whether confirmation is required.

This property is in the **Interaction** property group.

#### confirmText

Use this property to write your own text for the confirmation dialog. Otherwise, default text is used. See commandConfirm.

This property is in the **Interaction** property group.

#### drillDownColumnSubs

Use this property to direct a dashboard to assign data-table column values to specified dashboard variables when the end user activates a drilldown on this object. In the **Object Properties** window, double-click on `drillDownColumnSubs` in the **Property Name** field to bring up the **Drill Down Column Substitutions** dialog.

The dialog has the following fields and buttons:

- **Substitution String**: Enter the dashboard variable next to the name of the data table column whose value you want assigned to the variable. Press Enter.

- **Add Column**: Enter the name of a column and click the **Add Column** button to insert a column into the table.
- **Clear**: Click the **Clear** button to remove all variables listed.

The **Column Name** list is populated based on the table's data attachment. If you have not yet attached the table to data, this list is empty.

Once you have selected which column values to pass in as substitutions, double-click on any element in your object to open a drill down window that displays corresponding values.

This property is in the **Interaction** property group.

#### drillDownSelectMode

Use this property to control how a drill down display is activated. Select one of the following:

- **Anywhere** to activate a drill down display by double-clicking anywhere on the chart.
- **Element Only** to enable a drill down display only when you double-click on an element of the chart, such as a bar or candlestick.

This property is in the **Interaction** property group.

#### drillDownTarget

To specify a drill down display, double click on `drillDownTarget` in the **Property Name** field to bring up the **Drill Down Properties** dialog. See "Drill-Down Specification" on page 535.

This property is in the **Interaction** property group.

## Rotated table: Label group

Properties in this group control the table's main label, including text, alignment, color, font, and size.

#### Label group properties

The group contains the following properties:

- "label" on page 444
- "labelTextAlignX" on page 445
- "labelTextColor" on page 445
- "labelTextFont" on page 445
- "labelTextHeight" on page 445

#### label

Specifies the text for the chart label. Click the ellipsis for multi-line text.

The default is **Table**.

This property is in the **Label** property group.

#### labelTextAlignX

Sets the alignment of the chart label (see the label property). Select **Left**, **Center**, or **Right** from the drop down list.

This property is in the **Label** property group.

#### labelTextColor

Specifies the color of the chart label text (see the label property). Select the **...** button and choose a color from the palette. Close the Color Chooser window when you are done.

This property is in the **Label** property group.

#### labelTextFont

Specifies the font of the chart label text (see the label property). Select an item from drop down list.

This property is in the **Label** property group.

#### labelTextHeight

Specifies the point size of the chart label text (see the label property).

This property is in the **Label** property group.

## Rotated table: Object group

Properties in this group control the visibility and transparency of the table as a whole. They also control (or reflect) the overall position and dimensions of the table. In addition, a property in this group reflects the generated name of this individual table.

#### Object group properties

The group contains the following properties:

- "anchor" on page 446
- "dock" on page 446
- "objHeight" on page 446
- "objName" on page 446
- "objWidth" on page 446
- "objX" on page 446
- "objY" on page 446
- "visFlag" on page 447

#### anchor

Select zero or more of **Top**, **Left**, **Bottom**, and **Right** in order to control the object's placement. The `anchor` property is only applied when the display is resized either by changing the **Background Properties** on the display or by resizing the window in **Layout** mode. If an object has the `dock` property set, the `anchor` property is ignored. See "About resize modes" on page 39.

#### dock

Select **None** (default), **Top**, **Left**, **Bottom**, **Right**, or **Fill** in order to control the object's placement in **Layout** resize mode. See "About resize modes" on page 39.

#### objHeight

Set the height of a chart by entering a value for this property or by dragging a handle of the bounding box that appears when the chart is selected. When you drag a handle of the bounding box, the displayed value for this property changes to reflect the real-time height of the chart.

This property is in the **Object** property group.

#### objName

An identifier that is generated by the Dashboard Builder. This name can be used by other objects' properties in order to refer to the named chart.

This property is in the **Object** property group.

#### objWidth

Set the width of a chart by entering a value for this property or by dragging a handle of the bounding box that appears when the chart is selected. When you drag a handle of the bounding box, the displayed value for this property changes to reflect the real-time width of the chart.

This property is in the **Object** property group.

#### objX

Sets the X coordinate of the center of this visualization object, relative to the lower left corner of the current dashboard. This value is set automatically when you position the object with the mouse.

This property is in the **Object** property group.

#### objY

Sets the Y coordinate of the center of this visualization object, relative to the lower left corner of the current dashboard. This value is set automatically when you position the object with the mouse.

This property is in the **Object** property group.

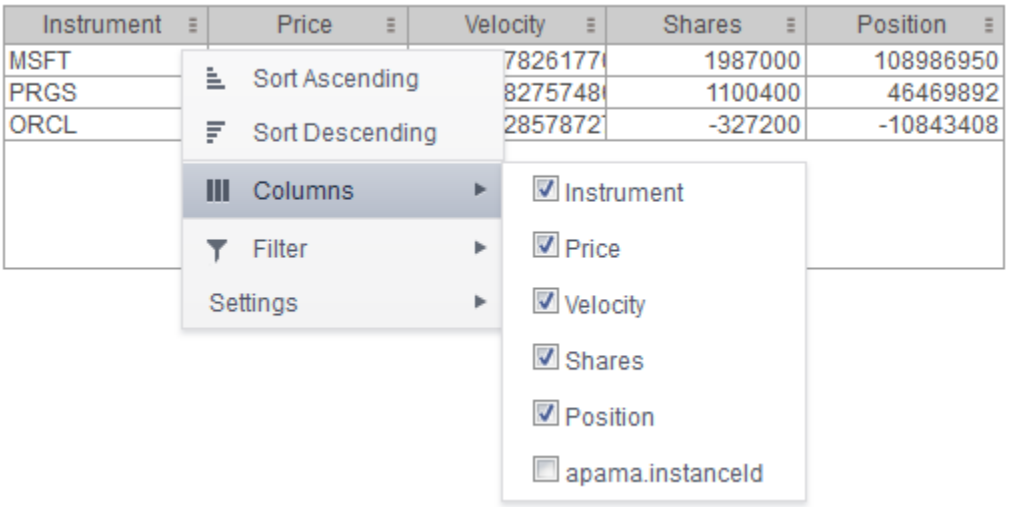
#### visFlag

Deselect to make this visualization object invisible in the current dashboard.

This property is in the **Object** property group.

## HTML5 tables

The HTML5 table is an advanced HTML implementation of obj_table02 which provides enhanced filtering. The HTML5 table is available in the Thin Client only.




To display your data in a HTML5 table, select the **Table (HTML5)** table object instance in the Builder's **Tables** tab. One can also select any of the other table objects under Tables tab except the **Rotated Table** and check the `webGridFlag` property.

#### Requirements

HTML5 table appears in the Thin Client in any modern version of a supported browser. No plug-in is required. In older browsers which do not support HTML5, the HTML5 table will be rendered with the classic grid table. In this release, the HTML5 table is not supported on iPad.

#### Features

The HTML5 table supports advanced, interactive table features in the Thin Client: sorting on multiple columns, filtering on multiple columns, column resizing, column reordering, and hiding columns. In addition, you can unsort a previously selected sort column and, in a grid with `rowHeaderEnabledFlag = true`, additional columns can be locked into the row header. You can save all of those column settings permanently so that they are restored when you return to the display later. Many of these features are accessed from the column menu, shown in the screen shot above, opened by clicking on the menu icon in each column's header.

Also, for improved performance and usability, if a data table contains more than 200 rows the HTML5 table displays it in pages of 200 rows, using a page control that appears at the bottom of the grid.

#### Column Sorting

Click on a column header to sort the table by that column. On the first click, the column is sorted in ascending order. On the second click the column is sorted in descending order. On the third click, the column is returned to its original unsorted state. A sort on a string column is case-insensitive.

You can select multiple sort columns. In that case, the sorting is performed in the order that the column headers were clicked. Multiple column sorting is a very useful feature, but can also cause confusion if you intend to sort on a single column, but forget to "unsort" any previously selected sort columns first. You should check for the up/down sort icon in other column headers if a sort gives unexpected results.

Column sorting is reflected in an export to HTML and Excel.

#### Column Visibility

You can hide or show columns in the table by clicking on any column's menu icon, and choosing **Columns** from the menu. This opens a submenu with a checkbox for each column that toggles the visibility of the column. All columns in the data table appear in the **Columns** menu, even those that are initially hidden by the obj_table02 property `columnsToHide`.

If the grid has the `rowHeaderEnabledFlag` property checked then the leftmost column (the row header column) cannot be hidden.

Column visibility changes are not reflected in an export to HTML and Excel.

#### Column Filtering

You can create a filter on any column. If filters are created on multiple columns, only the rows that pass all of the filters are displayed.

The background of a column's menu icon changes to white to indicate that a filter is defined on that column. This is intended to remind you which columns are filtered.

You can configure a filter on any column by clicking on the column's menu icon and choosing **Filter** from the menu. This opens the Column Filter dialog.

Options in the Column Filter dialog vary according to the data type of the selected column:

- **String columns**: You can enter a filter string such as "abc" and, from the dropdown list, select the operator (equal to, not equal to, starts with, contains, etc) to be used when comparing the filter string to each string in the column. All of the filter comparisons on strings are case-insensitive. You can optionally enter a second filter string (e.g. "xyz") and specify if an AND or OR combination should be used to combine the first and second filter results on the column.
- **Numeric columns**: You can enter numeric filter values and select arithmetic comparison operators, (=, !=, >, >=, <, <=). You can optionally enter a second filter value and

comparison operator, and specify if an AND or OR combination should be used to combine the first and second filter results.

- **Boolean columns**: You simply select whether matching items should be true or false.
- **Date columns**: You can select a date and time, and choose whether matching items should have a timestamp that is same as, before, or after the filter time. The date is selected by clicking the calendar icon and picking a date from the Calendar dialog. The time is selected by clicking the time icon and picking a time from the dropdown list. Alternatively, a date and time can be typed in the edit box.

Data updates to the grid are suspended while the filter menu is opened. The updates are applied when the menu is closed.

Column filtering is reflected in an export to HTML and Excel.

#### Column Locking

This feature is available only if the obj_table02 instance has the row header feature enabled (`rowHeaderEnabledFlag` is checked). If so, the leftmost column is "locked" in position, that is it does not scroll horizontally with the other columns in the table. If the row header is enabled, then two items labeled **Lock** and **Unlock** appear in the column menu. These can be used to add or remove additional columns from the non-scrolling row header area.

If the row header is enabled, at least one column must remain locked.

Column locking is not reflected in an export to HTML and Excel.

#### Column Reordering

You can reorder the grid columns by dragging and dropping a column's header into another position. If the grid has `rowHeaderEnabledFlag` checked, then dragging a column into or out of the row header area (the leftmost columns) is equivalent to locking or unlocking the column.

Column reordering is not reflected in an export to HTML and Excel.

#### Paging

If the data table contains more than one page of rows, the page controls are displayed at the bottom of the grid. The default page size is 200 but can be set on each obj_table02 instance via the new property named `webGridRowsPerPage`. The default value of that property is zero, which indicates that the default size (200) should be used. If the height of the grid is less than 64 pixels, there is insufficient space to display the page controls so only the rows on the first page will be viewable.

#### Row Mouseover

A new property named `webGridHoverColor` is available on obj_table02. It is visible only if `webGridFlag = true` . The default value of `webGridHoverColor` is checked. If it is set to any other color index value, then that color is used to highlight the row that is under the mouse cursor. But if the obj_table02 `filterProperties` feature is used to color rows, that color takes precedence, so the `webGridHoverColor` may not be useful in those cases. Also, if the row header is enabled, the row header column and the other

columns are highlighted separately, according to which section of the grid the mouse is over.

#### Saving Settings

You can permanently save all of the custom settings made to the grid, including filtering, sorting, column size (width), column order, column visibility, and column locking. This is done by opening any column menu, clicking **Settings** , and then clicking **Save All**.

The grid's settings are written as an item in the browser's local storage. The item's value is a string containing the grid's settings. The item uses a unique key comprised of the URL path name, the display name, and the obj_table02 instance's RTView object name. If the Thin Client's login feature is enabled, the key will also include the username and role, so different settings can be saved for each user and role for a grid on any given display, in the same browser and host.

If the user saves the grid settings and navigates away from the display or closes the browser, then the next time the user returns to the display in the same browser, the settings are retrieved from the browser's local storage and applied to the grid. The browser's local storage items are persistent, so the grid settings are preserved if the browser is closed and reopened or if the host system is restarted.

If the obj_table02 has `autoResizeFlag = true`, then the column widths are not restored from the saved settings, and the values computed by the auto-resize feature are used instead. This is by design.

You can delete the grid's item from local storage by clicking **Settings -> Clear All** in any column menu. This permanently deletes the saved settings for the grid and returns the grid to the state defined in the display file. Note that each browser has its own local storage on each host. The local storage items are not shared between browsers on the same host or on different hosts. So, if a user logs in as Joe with role = admin in Internet Explorer on host H1, and saves grid settings for display X, then those grid settings are restored each time a user logs in as Joe, role admin, on host H1 and opens display X in Internet Explorer. But if all the same is true except that the browser is Chrome, then the settings saved in Internet Explorer are not applied. Or if the user is Joe and role is admin and the browser is Internet Explorer and the display is X, but the host system is H2 not H1, then the grid settings saved on H1 are not applied.

#### Support for Large Tables

The HTML5 table can support data tables with many rows and columns. However, for best performance the display server's `cellsperpage` property should be specified so that the server sends large tables to the client in pages, rather than sending all of the rows. In this server paging mode, large tables are also filtered and sorted in the Display Server, to improve performance and decrease data traffic. See the RTView documentation for a description of the `cellsperpage` property, and the related `cellsperexport` and `cellsperreport` properties. A typical value for `cellsperpage` is 20000.

#### Unsupported obj_table02 Features

The following are existing features of obj_table02 that are not supported by the HTML5 table:

- The `rowHeaderEnabledFlag` property is supported, but `rowHeaderBgColor`, `rowHeaderTextColor`, `rowHeaderTextFont`, `rowHeaderTextSize` are ignored. Instead the row header column is rendered like all other columns.
- The `columnResizeEnabledFlag` is ignored if it is false, the HTML5 table always allows column resizing.
- The `editDataEnabledFlag` is ignored, the table editing feature using custom commands is currently not supported.

Other limitations, and differences between the new and classic grids:

- Time zones: The strings shown in a date column are formatted by the display server using its time zone. But if a filter is specified on a date column, the date and time for the filter are computed using the client system's time zone. This can be confusing if the display server and client are in different time zones.
- Selected rows: The grid's row selection is cleared if the sort is changed or if columns are resized or reordered.
- Scrollbars: In general the grid only displays scrollbars when they are needed. However, the HTML5 table and the classic grid use different algorithms for deciding when to show or hide scrollbars, and do not use identical row heights and column widths. So the HTML5 table may sometimes display scrollbars when the classic grid does not, for a grid instance with a given width and height.
- Keyboard traversal: In the classic grid, selecting a row and then using the up/down arrow keys changes the selection to the previous/next row. In the HTML5 table, the arrow keys moves the keyboard focus to another row, as indicated by a highlight border around the focused table cell, but the user must press the space bar to select the row that contains the highlighted (focused) cell.
- Column widths: On a HTML5 table with no locked columns (`rowHeaderEnabledFlag = false`), columns expand to fill any unused width in the table, even if `autoResizeFlag = false`. That is, if the total width of the columns is less than the grid width (ie. the columns don't use all of the available width) then each column is expanded proportionally to fill the table. In contrast, the classic and Swing (Viewer) grids just leave unused space at the right edge of the grid. If the grid has locked columns (`rowHeaderEnabledFlag = true`), then the HTML5 table behaves the same as the classic and Swing grids.
- Export: The export to HTML and export to excel features are supported on the HTML5 table, and behave much the same as on the classic grid. The exported table respects the grid's filter and sort settings but ignores any column reordering, sizing, or hiding changes made by the user.
- Data updates to the grid are suspended while the filter menu is opened. The updates are applied when the menu is closed.

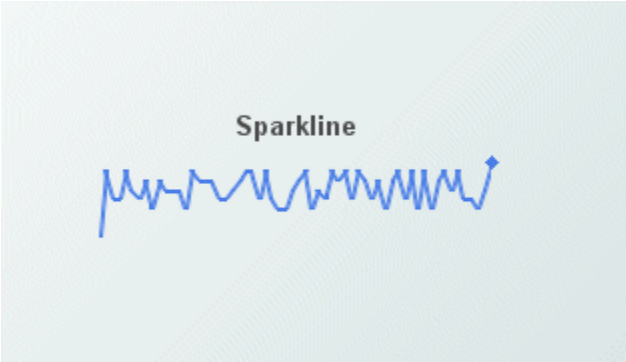
# 13 Trend Objects


- Sparkline charts ..... 454
- Stock charts ..... 472
- Trend graphs ..... 497

This chapter describes the visualization objects in the **Trends** tab of the Dashboard Builder tool.

## Sparkline charts

Sparkline charts are generally used to present trends and variations in a simple and condensed way. As the name implies there is a line associated with data, but no background or axis. It is possible to add labels at the beginning and ending points of the line, which then can be toggled on and off.




Attach scalar data to the `value` property or tabular data to the `valueTable` property. Tabular data attached to the `valueTable` property should have two columns: the first must contain numeric values or time stamps (x-axis values), and the second column should contain the corresponding (y-axis) numeric values.

When a sparkline chart is selected in the Builder canvas, the **Object Class Name** that appears at the top of the **Object Properties** pane is **obj_sparkline**.

The **Object Properties** panel organizes stock chart properties into the groups below.

### Sparkline chart: Alert group

Properties in this group allow you to set marker colors and styles based on a threshold value.

#### Alert group properties

The group contains the following properties:

- "valueHighAlarmEnabledFlag" on page 455
- "valueHighAlarm" on page 455
- "valueHighAlarmLineVisFlag" on page 455
- "valueHighAlarmMarkColor" on page 455
- "valueHighAlarmMarkStyle" on page 455
- "valueHighAlarmTraceColor" on page 455

- "valueHighAlarmTraceStyle" on page 456
- "valueLowAlarmEnabledFlag" on page 456
- "valueLowAlarm" on page 456
- "valueLowAlarmLineVisFlag" on page 456
- "valueLowAlarmMarkColor" on page 456
- "valueLowAlarmMarkStyle" on page 456
- "valueLowAlarmTraceColor" on page 457
- "valueLowAlarmTraceStyle" on page 457


#### valueHighAlarmEnabledFlag

Select to enable the high alarm threshold and related properties.

This property is in the **Alert** property group.

#### valueHighAlarm

Set the value of the high alarm threshold.

This property is in the **Alert** property group.

#### valueHighAlarmLineVisFlag

Select to display a dotted line at the high alarm threshold. The color of the line is set to the `valueHighAlarmMarkColor`.

This property is in the **Alert** property group.

#### valueHighAlarmMarkColor

When a trace marker's value is greater than or equal to the `valueHighAlarm` property, the marker will change to the `valueHighAlarmMarkColor` and `valueHighAlarmMarkStyle`.

This property is in the **Alert** property group.

#### valueHighAlarmMarkStyle

When a trace marker's value is greater than or equal to the `valueHighAlarm` property, the marker will change to the `valueHighAlarmMarkColor` and `valueHighAlarmMarkStyle`.

This property is in the **Alert** property group.

#### valueHighAlarmTraceColor

When the value of any segment of a trace line is greater than or equal to the `valueHighAlarm` property, that segment of the trace line will change to the `valueHighAlarmTraceColor` and `valueHighAlarmTraceStyle`.

**Note:** If `valueHighAlarmTraceStyle` is set to **No Line**, then `valueHighAlarmTraceColor` will not change.

This property is in the **Alert** property group.

#### valueHighAlarmTraceStyle

When the value of any segment of a trace line is greater than or equal to the `valueHighAlarm` property, that segment of the trace line will change to the `valueHighAlarmTraceColor` and `valueHighAlarmTraceStyle`.

**Note:** If `valueHighAlarmTraceStyle` is set to **No Line**, then `valueHighAlarmTraceColor` will not change.

This property is in the **Alert** property group.

#### valueLowAlarmEnabledFlag

Select to enable the low alarm threshold and related properties:

This property is in the **Alert** property group.

#### valueLowAlarm

Set the value of the low alarm threshold.

This property is in the **Alert** property group.

#### valueLowAlarmLineVisFlag

Select to display a dotted line at the low alarm threshold. The color of the line is set to the `valueLowAlarmMarkColor`.

This property is in the **Alert** property group.

#### valueLowAlarmMarkColor

When the trace marker's value is less than or equal to the `valueLowAlarm` property, the marker will change to the `valueLowAlarmMarkColor` and `valueLowAlarmMarkStyle`.

This property is in the **Alert** property group.

#### valueLowAlarmMarkStyle

When the trace marker's value is less than or equal to the `valueLowAlarm` property, the marker will change to the `valueLowAlarmMarkColor` and `valueLowAlarmMarkStyle`.

This property is in the **Alert** property group.

#### valueLowAlarmTraceColor

When the value of any segment of a trace line is less than or equal to the `valueLowAlarm` property, that segment of the trace line will change to the `valueLowAlarmTraceColor` and `valueLowAlarmTraceStyle`.

This property is in the **Alert** property group.

#### valueLowAlarmTraceStyle

When the value of any segment of a trace line is less than or equal to the `valueLowAlarm` property, that segment of the trace line will change to the `valueLowAlarmTraceColor` and `valueLowAlarmTraceStyle`.

This property is in the **Alert** property group.

## Sparkline chart: Background group

Properties in this group control the visibility and appearance of the portion of the graph that serves as the background of the plot area.

#### Background group properties

The group contains the following properties:


- "bgBorderColor" on page 457
- "bgBorderFlag" on page 458
- "bgColor" on page 458
- "bgEdgeWidth" on page 458
- "bgGradientColor2" on page 458
- "bgGradientMode" on page 458
- "bgRaisedFlag" on page 459
- "bgRoundness" on page 459
- "bgShadowFlag" on page 459
- "bgStyleFlag" on page 459
- "bgVisFlag" on page 459
- "borderPixels" on page 459


#### bgBorderColor

Sets the color of the border (see `bgBorderFlag`) of the background rectangle. Select the **...** button and choose a color from the palette. Close the Color Chooser window when you are done.

This property is in the **Background** property group.

#### bgBorderFlag

Select to display a border around the background rectangle.

This property is in the **Background** property group.

#### bgColor

Sets the background color. Select the **...** button and choose a color from the palette. Close the Color Chooser window when you are done.

This property is in the **Background** property group.

#### bgEdgeWidth

Sets the width in pixels of the 3D edge on the background rectangle. This property is only used if `bgBorderFlag` is selected.

This property is in the **Background** property group.

#### bgGradientColor2

Sets the color for the second color in the gradient. The default is white. The `bgColor` property sets the first color in the gradient.

This property is in the **Background** property group.

#### bgGradientMode

Display a gradient in the background rectangle. Select from the following options:

- **None**: No gradient.
- **Diagonal Edge**: Gradient is drawn at a 45 degree angle from the top left to the bottom right corner of the object.
- **Diagonal Center**: Gradient is drawn at a 45 degree angle from the center to the top left and the bottom right corners of the object.
- **Horizontal Edge**: Gradient is drawn horizontally from the top to the bottom of the object.
- **Horizontal Center**: Gradient is drawn horizontally from the center to the top and bottom of the object.
- **Vertical Edge**: Gradient is drawn vertically from the left to the right of the object.
- **Vertical Center**: Gradient is drawn vertically from the center to the left and right of the object.

This property is in the **Background** property group.

#### bgRaisedFlag

Reverses the direction of the gradient, as well as that of the 3D edge if the `bgStyle` selected is 3D Rectangle.

This property is in the **Background** property group.

#### bgRoundness

Sets the arc length of the rounded corners. This property is only available if the `bgStyle` selected is **Round Rectangle**.

The value of `bgRoundness` cannot exceed half the value of the `objWidth` or the `objHeight`. If `bgRoundness` does exceed that value, half of `objWidth` or `objHeight` (whichever is smaller) will be used instead. For example if `objWidth` is `100` and `objHeight` is `50`, then the value of `bgRoundness` cannot exceed `25`. If it does, then half the value of `objHeight` (`25`) will be used instead.

This property is in the **Background** property group.

#### bgShadowFlag

Select to display a drop shadow on the background rectangle.

This property is in the **Background** property group.

#### bgStyleFlag

Choose one of the following three options from the drop down menu:

- **Rectangle**: Select to display a background rectangle.
- **3D Rectangle**: Select to display a 3D edge on the background rectangle. If selected, use `bgEdgeWidth` to set the width of the 3D edge.
- **Round Rectangle**: Select to display a background rectangle with rounded edges. If selected, use `bgRoundness` to set the arc length of the rounded corners.

This property is in the **Background** property group.

#### bgVisFlag

Select to display the background rectangle.

This property is in the **Background** property group.

#### borderPixels

Sets the width in pixels of the border between the chart and the edge of the background rectangle.

This property is in the **Background** property group.

## Sparkline chart: Data group

Properties in this group control the data to which the chart is attached, as well as the y-axis range and the maximum number of data points contained in the chart.

### Data group properties

This group includes the following properties:


- "maxPointsPerTrace" on page 460
- "value" on page 460
- "valueDivisor" on page 460
- "valueTable" on page 460
- "yValueMax" on page 460
- "yValueMin" on page 461


#### maxPointsPerTrace

The default is 1000. The maximum value for this property is 30000.

This property is in the **Data** property group.

#### value

Attach your scalar data to the `value` property.

This property is in the **Data** property group.

#### valueDivisor

Divides y-axis values by the number entered.

This property is in the **Data** property group.

#### valueTable

Attach your tabular data to the `valueTable` property. Tabular data attached must have two columns: the first must contain numeric values or time stamps (x-axis values) and the second column should contain the corresponding (y-axis) numeric values.

This property is in the **Data** property group.

#### yValueMax

Controls the range of y-axis if the `yAxisAutoScaleMode` is set to **Off**. Select **On** for the `yAxisAutoScaleMode` to calculate the y-axis range according to data values being plotted. To calculate the y-axis range including `yValueMin` and `yValueMax`, select **On - Include Min/Max**.

This property is in the **Data** property group.

#### yValueMin

Controls the range of y-axis if the `yAxisAutoScaleMode` is set to **Off**. Select **On** for the `yAxisAutoScaleMode` to calculate the y-axis range according to data values being plotted. To calculate the y-axis range including `yValueMin` and `yValueMax`, select **On - Include Min/Max**.

This property is in the **Data** property group.

## Sparkline chart: Data Format group

This group contains the yValueFormat property, which controls the format of displayed values.

#### yValueFormat

Select or enter the numeric format of values displayed in the legend and popup legend. To enter a format, use syntax from the Java `DecimalFormat` class.

This property is in the **Data Format** property group.

## Sparkline chart: Historian group

Do not use the properties in this group.

#### historyTableName

Do not use this property.

#### historyTableRowNameFlag

Do not use this property.

## Sparkline chart: Interaction group

Properties in this group control various forms of interaction between the end user and the chart, including activating commands, drill downs, and tooltips.

#### Interaction group properties

The group includes the following properties:


- "command" on page 462
- "commandCloseWindowOnSuccess" on page 462
- "commandConfirm" on page 462
- "confirmText" on page 463
- "cursorColor" on page 463
- "cursorFlag" on page 463

- "drillDownTarget" on page 463
- "legendPopupFlag" on page 463


#### command

Assign a command or group of commands to this stock chart by right-clicking on the `command` property name in the Object Properties window. Select **Define Command** and choose **SYSTEM**, **APAMA**, or **MULTIPLE**. See "Using the Define Apama Command dialog" on page 226.

Once a command or command group has been assigned to this object, you can activate it from a deployed dashboard or from the Dashboard Builder:

- Dashboard Builder: Double click on the object.
- Web-based deployment: Single click on the object or else right click on it and select **Execute Command** from the popup menu.
- Local deployment: By default, single-click on the object or else right-click on it and select **Execute Command** from the popup menu. To override the default, select **Tools > Options** in the Builder (do this before you generate the deployment package), and uncheck **Single-Click for Drill Down and Commands** in the **General** tab. This allows the end user to use either a double click or a right click.

When you activate a command, any defined drill down substitutions are performed, and then the command is executed.

If you assign multiple commands, the commands are launched in an arbitrary order, and are executed asynchronously; there is no guarantee that one command will finish before the next one in the sequence starts.

This property is in the **Interaction** property group.

#### commandCloseWindowOnSuccess

Select this property to automatically close the window that initiates a **SYSTEM** command when the command is executed successfully. This applies to **SYSTEM** commands only, and is not supported at all for thin-client, Web-page deployments.

With **APAMA** commands, the window is closed whether or not the command is executed successfully. For **MULTIPLE** commands, the window closes when the first command in the command group succeeds.

This property is in the **Interaction** property group.

#### commandConfirm

By default, when the end user executes a command (see the `command` property), the command confirmation dialog is disabled. To control this option for each individual object, use the `commandConfirm` check box. If confirmation is required for a **MULTIPLE** command group, a single confirmation dialog is presented; if you confirm the execution, all individual commands in the group are executed with no further confirmation. If the you cancel the execution, none of the commands in the group is executed.

You can also override the confirmation status of individual objects with an application-wide policy. Select **Tools > Options** and choose from three confirmation values:

- **Do not confirm**: Indicates that no commands require confirmation (regardless of each object's confirmation status).
- **Confirm all**: Indicates that all commands require confirmation (regardless of each object's confirmation status).
- **Use object confirm flag** (default): Indicates that the confirmation status of each object will determine whether confirmation is required.

This property is in the **Interaction** property group.

#### confirmText

Use this property to write your own text for the confirmation dialog. Otherwise, default text is used. See `commandConfirm`.

This property is in the **Interaction** property group.

#### cursorColor

Sets the color of the cursor. Select the **...** button and choose a color from the palette. Close the Color Chooser window when you are done.

The default is yellow.

This property is in the **Interaction** property group.

#### cursorFlag

Select to enable the cursor. When the cursor is enabled, select the chart and point to a location on a trace to see a cursor line at that location and display the time and values of the trend line at the cursor line on the legend. Select the `legendPopupFlag` to display the legend along the cursor.

The cursor is disabled by default.

This property is in the **Interaction** property group.

#### drillDownTarget

To specify a drill down display, double click on `drillDownTarget` in the **Property Name** field to bring up the Drill Down Properties dialog. See "Drill-Down Specification" on page 535.

This property is in the **Interaction** property group.

#### legendPopupFlag

Controls whether a legend pops up when you mouse over the trend line.

This property is in the **Interaction** property group.

## Sparkline chart: Label group

Properties in this group control the graph's main label (which defaults to **Sparkline**), including text, alignment, color, font, and size.

### Label group properties

The group includes the following properties:


- "label" on page 464
- "labelMinTabWidth" on page 464
- "labelTextAlignX" on page 464
- "labelTextAlignY" on page 464
- "labelTextColor" on page 465
- "labelTextFont" on page 465
- "labelTextHeight" on page 465


#### label

Specifies the text for the chart label. Click the ellipsis for multi-line text.

The default is **Sparkline**.

This property is in the **Label** property group.

#### labelMinTabWidth

Sets minimum width of the label tab. This property only applies if `labelTextAlignY` is set to **TabTop**.

This property is in the **Label** property group.

#### labelTextAlignX

Sets the x-axis alignment of the chart label (see the `label` property). Select **Left**, **Center**, or **Right** from the drop down list.

This property is in the **Label** property group.

#### labelTextAlignY

Sets the y-axis position of the chart label (see the `label` property). Select one of the following from the drop down list:

- **Outside Top**: Well above the background rectangle
- **Top**: Just above the background rectangle
- **Title Top**: Along the top line of the background rectangle

- **Tab Top**: Just above the background rectangle. Height and width of the tab is dependent on the height and width of the text. Use the `labelMinTabWidth` property to specify a minimum tab width.
- **Inside Top**: Inside the top of the background rectangle

This property is in the **Label** property group.

#### labelTextColor

Specifies the color of the chart label text (see the `label` property). Select the **...** button and choose a color from the palette. Close the Color Chooser window when you are done.

This property is in the **Label** property group.

#### labelTextFont

Specifies the font of the chart label text (see the `label` property). Select an item from drop down list.

This property is in the **Label** property group.

#### labelTextHeight

Specifies the point size of the chart label text (see the `label` property).

This property is in the **Label** property group.

## Sparkline chart: Legend group

Properties in this group control the visibility, appearance, and content of the chart legend.

#### Legend group properties

The group contains the following properties:

- "legendBgColor" on page 465
- "legendBgGradientColor2" on page 466
- "legendBgGradientMode" on page 466
- "legendTimeFormat" on page 466
- "legendValueMinSpace" on page 466
- "legendVisFlag" on page 466
- "legendWidthPercent" on page 467

#### legendBgColor

Select the **...** button and choose a color from the palette to set the background color of the legend. Close the Color Chooser window when you are done.

This property is in the **Legend** property group.

#### legendBgGradientColor2

Sets the color for the second color in the gradient. The default is white. The `bgColor` property sets the first color in the gradient.

This property is in the **Background** property group.

#### legendBgGradientMode

Display a gradient in the legend background rectangle. Select from the following options:

- **None**: No gradient.
- **Diagonal Edge**: Gradient is drawn at a 45 degree angle from the top left to the bottom right corner of the object.
- **Diagonal Center**: Gradient is drawn at a 45 degree angle from the center to the top left and the bottom right corners of the object.
- **Horizontal Edge**: Gradient is drawn horizontally from the top to the bottom of the object.
- **Horizontal Center**: Gradient is drawn horizontally from the center to the top and bottom of the object.
- **Vertical Edge**: Gradient is drawn vertically from the left to the right of the object.
- **Vertical Center**: Gradient is drawn vertically from the center to the left and right of the object.

This property is in the **Background** property group.

#### legendTimeFormat

Sets the format for the time displayed in the legend. Use syntax from the Java `SimpleDateFormat` class. For example, **MMMM dd, yyyy hh:mm:ss** results in the form **August 30, 2010 05:32:12 PM**. If no format is given, the `timeFormat` is used.

This property is in the **Legend** property group.

#### legendValueMinSpace

Specify the minimum distance in pixels between values and labels in the legend.

This property is in the **Legend** property group.

#### legendVisFlag

Select to display the legend.

This property is in the **Legend** property group.

#### legendWidthPercent

Sets the percent of the total width of the object used for the legend.

This property is in the **Legend** property group.

## Sparkline chart: Marker group

This group contains the property markDefaultSize, which controls the size of the trace marker.

#### markDefaultSize

Sets the size, in pixels, of the marker at the end of the trace line. Supply an integer value that is between 1 and 18, inclusive.

This property is in the **Marker** property group.

## Sparkline chart: Object group

Properties in this group control the visibility and transparency of the chart as a whole. They also control (or reflect) the overall position and dimensions of the chart. In addition, a property in this group reflects the generated name of this individual chart.

#### Object group properties

This group contains the following properties:


- "anchor" on page 467
- "dock" on page 468
- "objHeight" on page 468
- "objName" on page 468
- "objWidth" on page 468
- "objX" on page 468
- "objY" on page 468
- "transparencyPercent" on page 468
- "visFlag" on page 469


#### anchor

Select zero or more of **Top**, **Left**, **Bottom**, and **Right** in order to control the object's placement. The `anchor` property is only applied when the display is resized either by changing the **Background Properties** on the display or by resizing the window in **Layout** mode. If an object has the `dock` property set, the `anchor` property is ignored. See "About resize modes" on page 39.

#### dock

Select **None** (default), **Top**, **Left**, **Bottom**, **Right**, or **Fill** in order to control the object's placement in **Layout** resize mode. See "About resize modes" on page 39.

#### objHeight

Sets the height of a chart by entering a value for this property or by dragging a handle of the bounding box that appears when the stock chart is selected. When you drag a handle of the bounding box, the displayed value for this property changes to reflect the real-time height of the chart.

This property is in the **Object** property group.

#### objName

An identifier that is generated by the Dashboard Builder. This name can be used by other objects' properties in order to refer to the named stock chart.

This property is in the **Object** property group.

#### objWidth

Sets the width of a chart by entering a value for this property or by dragging a handle of the bounding box that appears when the stock chart is selected. When you drag a handle of the bounding box, the displayed value for this property changes to reflect the real-time width of the chart.

This property is in the **Object** property group.

#### objX

Sets the X coordinate of the center of this visualization object, relative to the lower left corner of the current dashboard. This value is set automatically when you position the object with the mouse.

This property is in the **Object** property group.

#### objY

Sets the Y coordinate of the center of this visualization object, relative to the lower left corner of the current dashboard. This value is set automatically when you position the object with the mouse.

This property is in the **Object** property group.

#### transparencyPercent

Sets the transparency of this chart.

This property is in the **Object** property group.

#### visFlag

Deselect to make this visualization object invisible in the current dashboard.

This property is in the **Object** property group.

## Sparkline chart: Plot Area group

The property in this group, traceBgColor, controls the color of the plot area.

#### traceBgColor

To set the color of the plot area, select the **...** button and choose a color from the palette to set the background color. Close the Color Chooser window when you are done.

This property is in the **Plot Area** property group.

## Sparkline chart: Trace group

The properties control the visibility and appearance of the trace line.

#### Trace group properties

The group includes the following properties:

- "traceFillStyle" on page 469
- "traceLabel" on page 470
- "traceLineColor" on page 470
- "traceLineStyle" on page 470
- "traceLineThickness" on page 470
- "traceNMarkColor" on page 470
- "traceNMarkStyle" on page 470
- "traceNValueHistoryFlag" on page 471

#### traceFillStyle

Select one of the following fill styles for from the drop down menu:

- **Solid**
- **Transparent**
- **Gradient**
- **Transparent Gradient**
- **None**

The default setting is **None**.

This property is in the **Trace** property group.

#### traceLabel

Enter a label for the trace line. This label appears in the chart's legend.

This property is in the **Trace** property group.

#### traceLineColor

Sets the trace line color. Select the **...** button and choose a color from the palette. Close the Color Chooser window when you are done.

This property is in the **Trace** property group.

#### traceLineStyle

Select one of the following line styles for the trace line from the drop down menu:

- **No Line**
- **Solid**
- **Dotted**
- **Dashed**
- **Dot Dashed**

This property is in the **Trace** property group.

#### traceLineThickness

Select one of the following thickness specifications for the price trace line from the drop down menu:

- **Thin**
- **Medium**
- **Thick**

This property is in the **Trace** property group.

#### trace*N*MarkColor

Select the **...** button and choose a color from the palette to set the trace marker color. Close the Color Chooser window when you are done.

This property is in the **Trace** property group.

#### trace*N*MarkStyle

Sets the style of the marker used on the trace. Select one of the following items from the drop down menu:

- No Marker

- Dot
- +
- *
- o
- x
- Filled Circle
- Filled Diamond
- Filled Triangle
- Filled Square
- Filled Star

This property is in the **Trace** property group.

#### traceNValueHistoryFlag

Do not use this property.

This property is in the **Trace** property group.

## Sparkline chart: X-Axis group

Properties in this group control the range of the x-axis.

#### X-Axis group properties

The group includes the following properties:

- "timeRange" on page 471
- "timeRangeBegin" on page 472
- "timeRangeEnd" on page 472

#### timeRange

Sets the total amount of time, in seconds, plotted on the chart.

If `timeRange` is set to **-1**, the time range is determined by the first and last timestamp found in the attached data.

**Note:** `timeRange` is ignored if both `timeRangeBegin` and `timeRangeEnd` are set.

The default is -1.0.

This property is in the **X-Axis** property group.

#### timeRangeBegin

Sets the start time value of the data to be plotted on the chart. Following are the supported formats:

- **mm/dd/yyyy hh:mm:ss** (e.g., **01/16/2004 12:30:03**)
- **yyyy-mm-dd hh:mm:ss** (e.g., **2004-01-16 12:30:03**)
- The number of milliseconds since midnight, January 1, 1970 UTC

**Note:** If only the time is specified, the current date is used.

This property is in the **X-Axis** property group.

#### timeRangeEnd

Sets the end time value of the data to be plotted on the chart. Following are the supported formats are:

- **mm/dd/yyyy hh:mm:ss** (e.g., **01/16/2010 12:30:03**)
- **yyyy-mm-dd hh:mm:ss** (e.g., **2010-01-16 12:30:03**)
- The number of milliseconds since midnight, January 1, 1970 UTC

**Note:** If only the time is specified, the current date is used.

This property is in the **X-Axis** property group.

### Sparkline chart: Y-Axis group

This group contains the yAxisAutoScalMode property, which controls the range of the y-axis.

#### yAxisAutoScalMode

Select how the y-axis range is calculated from the drop down menu:

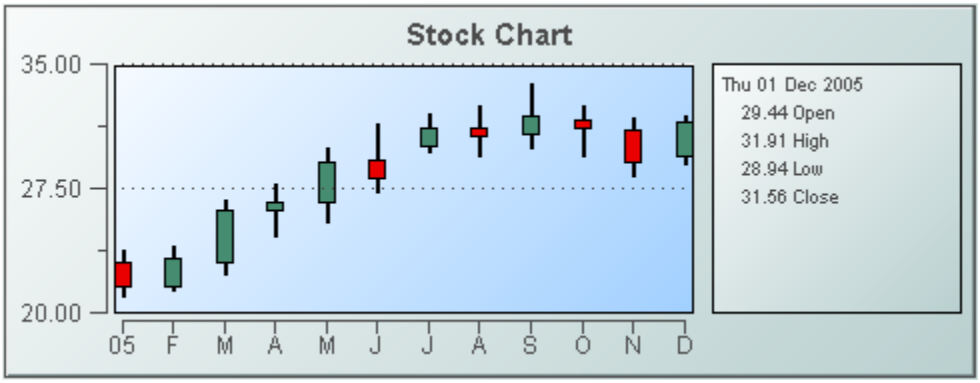
- **Off**: The `yValueMin` and `yValueMax` properties determine the range of y-axis.
- **On**: The dashboard calculates the y-axis range according to data values being plotted.
- **On - Include Min/Max**: The dashboard calculates the smallest range (with rounding) that includes `yValueMin` and `yValueMin` as well as all plotted points.

This property is in the **Y-Axis** property group.

## Stock charts

Stock charts visualize live and historical data related to financial instrument trades. They can include overlays that allow the display of data from multiple instruments or the display of periodic events such as stock splits and earnings announcements.

Each plotted point on a stock chart encapsulates four pieces of quantitative information for a particular instrument and time period: opening value, high value, low value, and closing value. Each chart visualizes tabular data that includes a time-valued column as well four numerical columns (for opening, high, low, and closing values).




Use the priceTraceCurrentTable and priceTraceHistoryTable properties to attach data to a stock chart. Use the timeRangeMode property to specify the duration of the time period represented by each plotted point.

Use the overlayCount property to specify the number of overlays to be included in the chart. Use the overlayNCurrentTable and overlayNHistoryTable properties to add the $N^{th}$ overlay.

When a stock chart is selected in the Builder canvas, the **Object Class Name** that appears at the top of the **Object Properties** pane is **obj_stockchart**.

The **Object Properties** panel organizes stock chart properties into the groups below.

## Stock chart: Background group

Properties in this group control the visibility and appearance of the portion of the chart that serves as the background of both the plot area and legend.

### Background group properties

The group contains the following properties:

- "bgBorderFlag" on page 474
- "bgColor" on page 474
- "bgEdgeWidth" on page 474
- "bgGradientColor2" on page 474
- "bgGradientMode" on page 474
- "bgRaisedFlag" on page 475
- "bgRoundness" on page 475
- "bgShadowFlag" on page 475

- "bgStyleFlag" on page 475
- "bgVisFlag" on page 475
- "borderPixels" on page 475


#### bgBorderFlag

Select to display a border around the background rectangle.

This property is in the **Background** property group.

#### bgColor

Sets the background color. Select the **...** button and choose a color from the palette. Close the Color Chooser window when you are done.

This property is in the **Background** property group.

#### bgEdgeWidth

Sets the width in pixels of the 3D edge on the background rectangle. This property is only used if `bgBorderFlag` is selected.

This property is in the **Background** property group.

#### bgGradientColor2

Sets the color for the second color in the gradient. The default is white. The `bgColor` property sets the first color in the gradient.

This property is in the **Background** property group.

#### bgGradientMode

Display a gradient in the background rectangle. Select from the following options:

- **None**: No gradient.
- **Diagonal Edge**: Gradient is drawn at a 45 degree angle from the top left to the bottom right corner of the object.
- **Diagonal Center**: Gradient is drawn at a 45 degree angle from the center to the top left and the bottom right corners of the object.
- **Horizontal Edge**: Gradient is drawn horizontally from the top to the bottom of the object.
- **Horizontal Center**: Gradient is drawn horizontally from the center to the top and bottom of the object.
- **Vertical Edge**: Gradient is drawn vertically from the left to the right of the object.
- **Vertical Center**: Gradient is drawn vertically from the center to the left and right of the object.

This property is in the **Background** property group.

#### bgRaisedFlag

Reverses the direction of the gradient, as well as that of the 3D edge if the `bgStyle` selected is 3D Rectangle.

This property is in the **Background** property group.

#### bgRoundness

Sets the arc length of the rounded corners. This property is only available if the `bgStyle` selected is **Round Rectangle**.

The value of `bgRoundness` cannot exceed half the value of the `objWidth` or the `objHeight`. If `bgRoundness` does exceed that value, half of `objWidth` or `objHeight` (whichever is smaller) will be used instead. For example if `objWidth` is `100` and `objHeight` is `50`, then the value of `bgRoundness` cannot exceed `25`. If it does, then half the value of `objHeight` (`25`) will be used instead.

This property is in the **Background** property group.

#### bgShadowFlag

Select to display a drop shadow on the background rectangle.

This property is in the **Background** property group.

#### bgStyleFlag

Choose one of the following three options from the drop down menu:

- **Rectangle**: Select to display a background rectangle.
- **3D Rectangle**: Select to display a 3D edge on the background rectangle. If selected, use `bgEdgeWidth` to set the width of the 3D edge.
- **Round Rectangle**: Select to display a background rectangle with rounded edges. If selected, use `bgRoundness` to set the arc length of the rounded corners.

This property is in the **Background** property group.

#### bgVisFlag

Select to display the background rectangle.

This property is in the **Background** property group.

#### borderPixels

Sets the width in pixels of the border between the chart and the edge of the background rectangle.

This property is in the **Background** property group.

## Stock chart: Data group

Properties in this group control the y-axis range.

### Data group properties

The group contains the following properties:

- "yValueMax" on page 476
- "yValueMin" on page 476

### yValueMax

Controls the y-axis range.

This property is in the **Data** property group.

### yValueMin

Controls the y-axis range.

This property is in the **Data** property group.

## Stock chart: Data Format group

The property in this group, yValueFormat, controls the numeric format of values displayed in the legend and popup legend.

### yValueFormat

Select or enter the numeric format of values displayed in the legend and popup legend. To enter a format, use syntax from the Java `DecimalFormat` class.

This property is in the **Data Format** property group.

## Stock chart: Interaction group

Properties in this group control various forms of interaction between the end user and the chart, including scrolling, zooming, and activating commands, drill downs, and tooltips.

### Interaction group properties

The group includes the following properties:

- "command" on page 477
- "commandCloseWindowOnSuccess" on page 477
- "commandConfirm" on page 478
- "confirmText" on page 478

- "cursorColor" on page 478
- "cursorFlag" on page 478
- "drillDownColumnSubs" on page 479
- "drillDownSelectMode" on page 479
- "drillDownTarget" on page 479
- "mouseOverFlag" on page 479
- "scrollbarMode" on page 480
- "scrollbarSize" on page 480
- "zoomEnabledFlag" on page 480


#### command

Assign a command or group of commands to this stock chart by right-clicking on the `command` property name in the Object Properties window. Select **Define Command** and choose **SYSTEM**, **APAMA**, or **MULTIPLE**. See "Using the Define Apama Command dialog" on page 226.

Once a command or command group has been assigned to this object, you can activate it from a deployed dashboard or from the Dashboard Builder:

- Dashboard Builder: Double click on the object.
- Web-based deployment: Single click on the object or else right click on it and select **Execute Command** from the popup menu.
- Local deployment: By default, single-click on the object or else right-click on it and select **Execute Command** from the popup menu. To override the default, select **Tools > Options** in the Builder (do this before you generate the deployment package), and uncheck **Single-Click for Drill Down and Commands** in the **General** tab. This allows the end user to use either a double click or a right click.

When you activate a command, any defined drill down substitutions are performed, and then the command is executed.

If you assign multiple commands, the commands are launched in an arbitrary order, and are executed asynchronously; there is no guarantee that one command will finish before the next one in the sequence starts.

This property is in the **Interaction** property group.

#### commandCloseWindowOnSuccess

Select this property to automatically close the window that initiates a **SYSTEM** command when the command is executed successfully. This applies to **SYSTEM** commands only, and is not supported at all for thin-client, Web-page deployments.

With **APAMA** commands, the window is closed whether or not the command is executed successfully. For **MULTIPLE** commands, the window closes when the first command in the command group succeeds.

This property is in the **Interaction** property group.

#### commandConfirm

By default, when the end user executes a command (see the command property), the command confirmation dialog is disabled. To control this option for each individual object, use the `commandConfirm` check box. If confirmation is required for a **MULTIPLE** command group, a single confirmation dialog is presented; if you confirm the execution, all individual commands in the group are executed with no further confirmation. If the you cancel the execution, none of the commands in the group is executed.

You can also override the confirmation status of individual objects with an application-wide policy. Select **Tools > Options** and choose from three confirmation values:

- **Do not confirm**: Indicates that no commands require confirmation (regardless of each object's confirmation status).
- **Confirm all**: Indicates that all commands require confirmation (regardless of each object's confirmation status).
- **Use object confirm flag** (default): Indicates that the confirmation status of each object will determine whether confirmation is required.

This property is in the **Interaction** property group.

#### confirmText

Use this property to write your own text for the confirmation dialog. Otherwise, default text is used. See commandConfirm.

This property is in the **Interaction** property group.

#### cursorColor

Sets the color of the cursor, as well as the zoom-area rectangle (see zoomEnabledFlag). Select the **...** button and choose a color from the palette. Close the Color Chooser window when you are done.

The default is yellow.

This property is in the **Interaction** property group.

#### cursorFlag

Select to enable the cursor. When the cursor is enabled, select the chart and point to a location on a trace to see a cursor line at that location and display the time and values of all traces at the cursor line on the legend. Select the legendPopupFlag to display the legend along the cursor.

The cursor is disabled by default.

This property is in the **Interaction** property group.

#### drillDownColumnSubs

Use this property to direct a dashboard to assign data-table column values to specified dashboard variables when the end user activates a drilldown on this object. In the Object Properties window, double-click on `drillDownColumnSubs` in the **Property Name** field to bring up the Drill Down Column Substitutions dialog.

The dialog has the following fields and buttons:

- **Substitution String**: Enter the dashboard variable next to the name of the data table column whose value you want assigned to the variable. Press Enter.
- **Add Column**: Enter the name of a column and click the **Add Column** button to insert a column into the table.
- **Clear**: Click the **Clear** button to remove all variables listed.

The **Column Name** list is populated based on the table's data attachment. If you have not yet attached the table to data, this list is empty.

Once you have selected which column values to pass in as substitutions, double-click on any element in your object to open a drill down window that displays corresponding values.

This property is in the **Interaction** property group.

#### drillDownSelectMode

Use this property to control how a drill down display is activated. Select one of the following:

- **Anywhere** to activate a drill down display by double-clicking anywhere on the chart.
- **Element Only** to enable a drill down display only when you double-click on an element of the chart, such as a bar or candlestick.

This property is in the **Interaction** property group.

#### drillDownTarget

To specify a drill down display, double click on `drillDownTarget` in the **Property Name** field to bring up the Drill Down Properties dialog. See "Drill-Down Specification" on page 535.

This property is in the **Interaction** property group.

#### mouseOverFlag

Select to enable trace element tooltips. When the enabled, hold the mouse over a location to display in a tooltip the time and the open and close values of all traces at the location.

This property is in the **Interaction** property group.

#### scrollbarMode

Sets whether and when the scroll bar appears in the chart. Select one of the following from the drop down menu:

- **Never**: Default setting
- **Always**: Display a scroll bar at all times.
- **As Needed**: Display the scroll bar when necessitated by zooming in the trace area, or when more data is loaded into the chart than is displayed in the time range. For example, if the time range of the data in your data attachment is greater than timeRange, setting `scrollbarMode` to **As Needed** will enable a scroll bar, allowing the end user to view all data loaded into the chart.

This property is in the **Interaction** property group.

#### scrollbarSize

Specifies the height of the horizontal scroll bar and the width of the vertical scroll bar, in pixels. The default value is -1, which sets the size to the system default.

This property is in the **Interaction** property group.

#### zoomEnabledFlag

Select to enable zooming within the chart. Click in the chart's trace area and drag the cursor until a desired range is selected. While dragging, a rectangle is drawn to show the zoom area. The rectangle's default color is yellow (this can be changed in the cursorColor property). After the zoom is performed, the chart stores up to four zoom operations in queue. To zoom out, press the shift key and click in the chart's trace area.

This property is in the **Interaction** property group.

## Stock chart: Historian group

Do not use the properties in this group.

#### historyTableName

Do not use this property.

#### historyTableRowNameFlag

Do not use this property.

## Stock chart: Label group

Properties in this group control the chart's main label (which defaults to **Stock Chart**), including text, alignment, color, font, and size.

#### Label group properties

The group includes the following properties:


- "label" on page 481
- "labelTextAlignX" on page 481
- "labelTextColor" on page 481
- "labelTextFont" on page 481
- "labelTextHeight" on page 481


#### label

Specifies the text for the chart label. Click the ellipsis for multi-line text.

The default is **Stock Chart**.

This property is in the **Label** property group.

#### labelTextAlignX

Specifies the alignment of the chart label. Select **Left**, **Right**, or **Center** from the drop down list.

This property is in the **Label** property group.

#### labelTextColor

Specifies the color of the chart label text. Select the **...** button and choose a color from the palette. Close the Color Chooser window when you are done.

This property is in the **Label** property group.

#### labelTextFont

Specifies the font of the chart label text. Select an item from drop down list.

This property is in the **Label** property group.

#### labelTextHeight

Specifies the point size of the chart label text.

This property is in the **Label** property group.

## Stock chart: Legend group

Properties in this group control the visibility, appearance, and content of the chart legend.

#### Legend group properties

The group contains the following properties:

- "legendBgColor" on page 482
- "legendBgGradientFlag" on page 482
- "legendPopupFlag" on page 482
- "legendValueMinSpace" on page 482
- "legendValueVisFlag" on page 482
- "legendVisFlag" on page 482
- "legendWidthPercent" on page 482


#### legendBgColor

Select the **...** button and choose a color from the palette to set the background color of the legend. Close the Color Chooser window when you are done.

This property is in the **Legend** property group.

#### legendBgGradientFlag

Select to display a gradient in the legend background.

This property is in the **Legend** property group.

#### legendPopupFlag

When the cursorFlag property is enabled, select `legendPopupFlag` to display the legend along the cursor.

This property is in the **Legend** property group.

#### legendValueMinSpace

Specifies the minimum distance in pixels between values and labels in the legend.

This property is in the **Legend** property group.

#### legendValueVisFlag

Select to display the numerical values of your data in the legend. If cursorFlag is enabled, the numerical values are always shown in the legend.

This property is in the **Legend** property group.

#### legendVisFlag

Select to display the legend.

This property is in the **Legend** property group.

#### legendWidthPercent

Sets the percent of the total width of the object used for the legend.

This property is in the **Legend** property group.

## Stock chart: Object group

Properties in this group control the visibility and transparency of the chart as a whole. They also control (or reflect) the overall position and dimensions of the chart. In addition, a property in this group reflects the generated name of this individual chart.

### Object group properties

This group contains the following properties:


- "anchor" on page 483
- "dock" on page 483
- "objHeight" on page 483
- "objName" on page 484
- "objWidth" on page 484
- "objX" on page 484
- "objY" on page 484
- "transparencyPercent" on page 484
- "visFlag" on page 484


#### anchor

Select zero or more of **Top**, **Left**, **Bottom**, and **Right** in order to control the object's placement. The `anchor` property is only applied when the display is resized either by changing the **Background Properties** on the display or by resizing the window in **Layout** mode. If an object has the `dock` property set, the `anchor` property is ignored. See "About resize modes" on page 39.

#### dock

Select **None** (default), **Top**, **Left**, **Bottom**, **Right**, or **Fill** in order to control the object's placement in **Layout** resize mode. See "About resize modes" on page 39.

#### objHeight

Set the height of a chart by entering a value for this property or by dragging a handle of the bounding box that appears when the chart is selected. When you drag a handle of the bounding box, the displayed value for this property changes to reflect the real-time height of the chart.

This property is in the **Object** property group.

#### objName

An identifier that is generated by the Dashboard Builder. This name can be used by other objects' properties in order to refer to the named chart.

This property is in the **Object** property group.

#### objWidth

Set the width of a chart by entering a value for this property or by dragging a handle of the bounding box that appears when the chart is selected. When you drag a handle of the bounding box, the displayed value for this property changes to reflect the real-time width of the chart.

This property is in the **Object** property group.

#### objX

Sets the X coordinate of the center of this visualization object, relative to the lower left corner of the current dashboard. This value is set automatically when you position the object with the mouse.

This property is in the **Object** property group.

#### objY

Sets the Y coordinate of the center of this visualization object, relative to the lower left corner of the current dashboard. This value is set automatically when you position the object with the mouse.

This property is in the **Object** property group.

#### transparencyPercent

Sets the transparency of this chart.

This property is in the **Object** property group.

#### visFlag

Deselect to make this visualization object invisible in the current dashboard.

This property is in the **Object** property group.

## Stock chart: Plot Area group

Properties in this group control the appearance of the plot area, the rectangular area that serves as background for the trace markers (but not for the legend or axis labels; see "Stock chart: Background group" on page 473). There is also a property that controls the color of the horizontal grid line or lines.

#### Plot Area group properties

The group includes the following properties:

- "gridBgColor" on page 485
- "gridBgGradientFlag" on page 485
- "gridBgImage" on page 485
- "gridColor" on page 485

#### gridBgColor

To set the color of the plot area, select the **...** button and choose a color from the palette to set the background color. Close the Color Chooser window when you are done.

This property is in the **Plot Area** property group.

#### gridBgGradientFlag

Select to display a gradient in the plot area.

This property is in the **Plot Area** property group.

#### gridBgImage

Specify an image (`.gif`, `.jpg`, or `.png` file) to display in the plot area. Select the name of the image file from the drop down menu, or enter the pathname of the file. The drop down menu contains the names of image files located in the current directory (typically, the `dashboards` directory of your project directory, under your Apama installation's work directory), as well as image files located in the first level of subdirectories. If you enter a pathname, use an absolute pathname or a pathname that is relative to the current directory.

This property is in the **Plot Area** property group.

#### gridColor

Sets the color of the dotted, horizontal grid line in the plot area (see xAxisMajorDivisions). Select the **...** button and choose a color from the palette. Close the Color Chooser window when you are done.

This property is in the **Plot Area** property group.

## Stock chart: Price Trace group

The properties control the visibility and appearance of the price trace, as well as the data to which it is attached.

#### Trace group properties

The group includes the following properties:

- "priceTraceBarGainColor" on page 486
- "priceTraceBarLossColor" on page 486
- "priceTraceCurrentTable" on page 486
- "priceTraceFillStyle" on page 487
- "priceTraceHistoryTable" on page 487
- "priceTraceLabel" on page 488
- "priceTraceLineColor" on page 488
- "priceTraceLineStyle" on page 488
- "priceTraceLineThickness" on page 488
- "priceTraceType" on page 489
- "priceTraceVisFlag" on page 489


#### priceTraceBarGainColor

Sets the color to indicate that a stock price value at market close is greater than value at market open. Select the **...** button and choose a color from the palette. Close the Color Chooser window when you are done.

The default is green.

> **Note:** This property does not apply if you have chosen **Line** for the priceTraceType property or both **Candlestick** for priceTraceType and **None** for priceTraceFillStyle.

This property is in the **Price Trace** property group.

#### priceTraceBarLossColor

Sets the color to indicate that a stock price value at market close is less than value at market open. Select the **...** button and choose a color from the palette. Close the Color Chooser window when you are done.

The default is red.

> **Note:** This property does not apply if you have chosen **Line** for the priceTraceType property or both **Candlestick** for priceTraceType and **None** for priceTraceFillStyle.

This property is in the **Price Trace** property group.

#### priceTraceCurrentTable

Attach your tabular data to the priceTraceHistoryTable and `priceTraceCurrentTable` properties. The `priceTraceCurrentTable` property is used for viewing live data. The

table in your data attachment should contain a single row that corresponds to and continually updates the last point on the graph.

Unless you attach this property to a scenario OHLC table, the table in your data attachment must contain the following five columns in this specific order:

- **Date**: Following are the supported formats for this column are:
  - `mm/dd/yyyy hh:mm:ss` (for example, **01/16/2010 12:30:03**)
  - `yyyy-mm-dd hh:mm:ss` (for example, **2010-01-16 12:30:03**)
  - The number of milliseconds since midnight, January 1, 1970 UTC
- **Open**: Value of stock price at first market open for defined time period
- **High**: High value of stock price for defined time period
- **Low**: Low value of stock price for defined time period
- **Close**: Value of stock price at last market close for defined time period

See "Attaching Dashboards to Correlator Data" on page 55.

This property is in the **Price Trace** property group.

#### priceTraceFillStyle

Select one of the following candlestick fill styles for from the drop down menu:

- **Solid**
- **Transparent**
- **Gradient**
- **Transparent Gradient**
- **None**

This setting has an effect only if priceTraceType is set to **CandleStick**.

The default setting is **None**.

This property is in the **Price Trace** property group.

#### priceTraceHistoryTable

Attach your tabular data to the `priceTraceHistoryTable` or priceTraceCurrentTable property. The `priceTraceHistoryTable` property is used for viewing and analyzing historical data (data generated before the correlator first sends data to this particular chart). The table in your data attachment should contain multiple rows, each corresponding to a point on the graph.

Unless you attach this property to a scenario OHLC table, the table in your data attachment must contain the following five columns in this specific order:

- **Date**: Following are the supported formats for this column:

- `mm/dd/yyyy hh:mm:ss` (for example, **01/16/2010 12:30:03**)
- `yyyy-mm-dd hh:mm:ss` (for example, **2010-01-16 12:30:03**)
- The number of milliseconds since midnight, January 1, 1970 UTC

- **Open**: Value of stock price at first market open for the defined time period
- **High**: High value of stock price for the defined time period
- **Low**: Low value of stock price for the defined time period
- **Close**: Value of stock price at last market close for the defined time period

See "Attaching Dashboards to Correlator Data" on page 55.

This property is in the **Price Trace** property group.

#### priceTraceLabel

Enter a label for the price trace line. This label appears in the chart's legend, as well as in the tooltip enabled by the mouseOverFlag property.

This property is in the **Price Trace** property group.

#### priceTraceLineColor

Sets the price trace line color. Select the **...** button and choose a color from the palette. Close the Color Chooser window when you are done.

**Note:** This property does not apply if you chose **OHLC** or **Bar** for priceTraceType.

This property is in the **Price Trace** property group.

#### priceTraceLineStyle

Select one of the following line styles for the price trace line from the drop down menu:

- **No Line**
- **Solid**
- **Dotted**
- **Dashed**
- **Dot Dashed**

This property is in the **Price Trace** property group.

#### priceTraceLineThickness

Select one of the following thickness specifications for the price trace line from the drop down menu:

- **Thin**

- **Medium**
- **Thick**

**Note:** This property does not apply if you chose **OHLC** or **Candlestick** for priceTraceType.

This property is in the **Price Trace** property group.

#### priceTraceType

Select one of the following trace types from the drop down menu:

- **Line**: A line graph that shows closing price values
- **Bar**: A bar graph that shows closing price values
- **OHLC**: A bar extending from the low to high price for each trading day. A left flange indicates the opening price and a right flange indicates the closing price. The priceTraceBarLossColor and priceTraceBarGainColor properties show whether the stock closed at a higher or lower price than the opening price.
- **Candlestick**: A bar extending from the opening to closing price for each trading period. The wicks on either end show the high and low for the trading period. The priceTraceBarLossColor and priceTraceBarGainColor properties show whether the stock closed at a higher or lower price than the opening price.

This property is in the **Price Trace** property group.

#### priceTraceVisFlag

Use the checkbox to control price trace visibility.

This property is in the **Price Trace** property group.

## Stock chart: Trace group

Properties in this group control the number of overlays the chart contains, as well as the overlay fill style.

#### Trace group properties

The group includes the following properties:


- "overlayCount" on page 489
- "overlayFillStyle" on page 490


#### overlayCount

Sets the number of overlays. The maximum is nineteen. For each overlay, the Dashboard Builder automatically creates a set of properties in the Object Properties window.

This property is in the **Trace** property group.

#### overlayFillStyle

When the value of overlayNType is **Line**, this specifies the effect with which to fill the area from the line to the bottom of the graph. The color is determined by overlayNLineColor. Select one of the following fill styles from the drop down menu:

- **Solid**
- **Transparent**
- **Gradient**
- **Transparent Gradient**.
- **None** (default)

This property is in the **Trace** property group.

## Stock chart: Trace*N* group

There is one group of these properties for each overlay in the chart (see overlayCount). The properties control the visibility and appearance of overlays, as well as the data to which they are attached.

#### Trace*N* group properties

The group includes the following properties:


- "overlayNCurrentTable" on page 490
- "overlayNHistoryTable" on page 491
- "overlayNLabel" on page 491
- "overlayNLineColor" on page 491
- "overlayNLineStyle" on page 491
- "overlayNLineThickness" on page 491
- "overlayNType" on page 492
- "overlayNVisFlag" on page 492


#### overlay*N*CurrentTable

The `overlayNCurrentTable` and overlayNHistoryTable properties are used in conjunction with the priceTraceHistoryTable or priceTraceCurrentTable properties to compare data (for example, to compare the activity of several stocks). See "Attaching Dashboards to Correlator Data" on page 55. To enable, set the overlayCount to the number of overlay traces that you want to show.

This property is in the **Trace***N* property group.

#### overlay*N*HistoryTable

The overlayNCurrentTable and `overlayNHistoryTable` properties are used in conjunction with the priceTraceHistoryTable or priceTraceCurrentTable properties to compare data (e.g. to compare the activity of several stocks). See "Attaching Dashboards to Correlator Data" on page 55. By default the overlays are disabled. To enable, set the overlayCount to the number of overlay traces you want to show.

This property is in the **Trace***N* property group.

#### overlay*N*Label

Enter a label for the overlay line. This label appears in legend and tooltip enabled by mouseOverFlag.

This property is in the **Trace***N* property group.

#### overlay*N*LineColor

Select the **...** button and choose a color from the palette to set the overlay line color. Close the Color Chooser window when you are done.

This property is in the **Trace***N* property group.

#### overlay*N*LineStyle

Select one of the following styles for the overlay line from the drop down menu:

- **No Line**
- **Solid**
- **Dotted**
- **Dashed**
- **Dot Dashed**

> **Note:** This property does not apply if you chose **Bar** or **Event** for overlayNType.

This property is in the **Trace***N* property group.

#### overlay*N*LineThickness

Select the following thickness of the overlay line from the drop down menu:

- **Thin**
- **Medium**
- **Thick**

> **Note:** This property does not apply if you chose **Bar** or **Event** for overlayNType.

This property is in the **Trace***N* property group.

#### overlay*N*Type

Select one of the following overlay types from the drop down menu:

- **Line**: A line graph that shows closing price values
- **Bar**: A bar graph that shows closing price values
- **Event**: A series of markers representing company events such as stock splits, company merges, etc. The first letter of the overlayNLabel is the letter that appears in each event marker.

This property is in the **Trace***N* property group.

#### overlay*N*VisFlag

Use the checkbox to control overlay visibility.

This property is in the **Trace***N* property group.

## Stock chart: X-Axis group

Properties in this group control the range and labeling of the x-axis, as well as the time interval between plotted points.

#### X-Axis group properties

The group includes the following properties:


- "timeFormat" on page 493
- "timeRange" on page 493
- "timeRangeBegin" on page 493
- "timeRangeEnd" on page 493
- "timeRangeMode" on page 494
- "tradeDayBegin" on page 494
- "tradeDayEnd" on page 494
- "tradeDayEndLabelFlag" on page 495
- "xAxisFlag" on page 495
- "xAxisLabel" on page 495
- "xAxisLabelTextHeight" on page 495
- "xAxisMajorDivisions" on page 495
- "xAxisMinorDivisions" on page 495

#### timeFormat

Sets the format for the time displayed in the x-axis using syntax from the Java `SimpleDateFormat` class. This property is only used when the timeRangeMode is **Continuous**.

For example, **MMMM dd, yyyy hh:mm:ss a** results in dates of the form **August 30, 2010 05:32:12 PM**. If no format is given, the date and time are not displayed on the x-axis.

Include a new line character (**'\n'**) to display multiple-line text in the time axis labels. For example, **MM\dd'\n'hh:mm:ss** results in the following form:

```
08\30
05:32:12
```

If left blank, the axis is labeled with a default format based on the range.

This property is in the **X-Axis** property group.

#### timeRange

Sets the total amount of time, in seconds, plotted on the chart.

If `timeRange` is set to **-1**, the time range is determined by the first and last timestamp found in the priceTraceHistoryTable and priceTraceCurrentTable. If both tables are empty, the chart uses the first and last timestamp of the first overlay trace that has a non-empty overlayNHistoryTable or overlayNCurrentTable.

**Note:** `timeRange` is ignored if both timeRangeBegin and timeRangeEnd are set.

The default is -1.0.

This property is in the **X-Axis** property group.

#### timeRangeBegin

Sets the start time value of the data to be plotted on the chart. Following are the supported formats:

- **mm/dd/yyyy hh:mm:ss** (e.g., **01/16/2010 12:30:03**)
- **yyyy-mm-dd hh:mm:ss** (e.g., **2010-01-16 12:30:03**)
- The number of milliseconds since midnight, January 1, 1970 UTC

**Note:** If only the time is specified, the current date is used.

This property is in the **X-Axis** property group.

#### timeRangeEnd

Sets the end time value of the data to be plotted on the chart. Following are the supported formats are:

- **mm/dd/yyyy hh:mm:ss** (e.g., **01/16/2010 12:30:03**)

- **yyyy-mm-dd hh:mm:ss** (e.g., **2010-01-16 12:30:03**)
- The number of milliseconds since midnight, January 1, 1970 UTC

**Note:** If only the time is specified, the current date is used.

This property is in the **X-Axis** property group.

#### timeRangeMode

Select the `timeRangeMode` from the drop down menu. This property sets the interval between trace data points. `timeRangeMode` also affects the x-axis labels. With some time intervals, for example, x-axis labels are dates, while with other time intervals, x-axis labels are times. There are eight modes:

- **Auto**: Selects the setting that best matches the time intervals in the price trace data table.
- **Intra-Day**: Time intervals are less than one day, for example, hourly or every 15 minutes.
- **Daily:** Time intervals are days.
- **Weekly**: Time intervals are weeks.
- **Monthly**: Time intervals are months.
- **Quarterly**: Time intervals are quarters.
- **Yearly**: Time intervals are annual.
- **Continuous**: Plots each point using the corresponding timestamp from the data table. This data can vary in time intervals.

**Note:** If the price trace data is more granular than the time interval specified in your data attachment, the price trace data will be aggregated to match the `timeRangeMode`.

This property is in the **X-Axis** property group.

#### tradeDayBegin

Defines the daily start time of the trading day. This property is used only with intraday data (time intervals less than one day, for example, hourly or every 15 minutes). The default value is `09:30`.

This property is in the **X-Axis** property group.

#### tradeDayEnd

Defines the daily end time of the trading day. This property is used only with intraday data (time intervals less than one day, for example, hourly or every 15 minutes). The default value is `16:00`.

This property is in the **X-Axis** property group.

#### tradeDayEndLabelFlag

Select to show the last data point of a day and the first data point of the next day (which are equal values) with separate points on the chart. Otherwise, they are shown together at one point on the chart.

This property is only used with intraday data.

The default is disabled.

This property is in the **X-Axis** property group.

#### xAxisFlag

Select to display the x-axis.

This property is in the **X-Axis** property group.

#### xAxisLabel

Specifies a label to display below the x-axis.

This property is in the **X-Axis** property group.

#### xAxisLabelTextHeight

Specifies the height in pixels of the x-axis labels.

This property is in the **X-Axis** property group.

#### xAxisMajorDivisions

Specify the number of major divisions on the x-axis.

This property is in the **X-Axis** property group.

#### xAxisMinorDivisions

Specify the number of minor divisions on the x-axis.

> **Note:** This property applies when the timeRangeMode property is set to **Continuous**.

This property is in the **X-Axis** property group.

## Stock chart: Y-Axis group

Properties in this group control the visibility and scaling of the y-axis or y-axes, as well as y-axis labeling and y-axis divisions. They also control the visibility of y-axis grid lines (but see also "Stock chart: Plot Area group" on page 484).

#### Y-Axis group properties

The group includes the following properties:


- "yAxisAutoScaleMode" on page 496
- "yAxisFlag" on page 496
- "yAxisFormat" on page 496
- "yAxisLabel" on page 496
- "yAxisLabelTextHeight" on page 497
- "yAxisMajorDivisions" on page 497
- "yAxisMinLabelWidth" on page 497
- "yAxisMinorDivisions" on page 497
- "yAxisMultiRangeFlag" on page 497
- "yAxisPercentFlag" on page 497


#### yAxisAutoScaleMode

Select how the y-axis range is calculated from the drop down menu:

- **Off**: The yValueMin and yValueMax properties determine the range of y-axis.
- **On**: The dashboard calculates the y-axis range according to data values being plotted.
- **On - Include Min/Max**: The dashboard calculates the smallest range (with rounding) that includes yValueMin and yValueMax as well as all plotted points.

This property is in the **Y-Axis** property group.

#### yAxisFlag

Select to display the y-axis.

This property is in the **Y-Axis** property group.

#### yAxisFormat

Select or enter the numeric format of values displayed on the y-axis. To enter a format, use syntax from the Java `DecimalFormat` class.

This property is in the **Y-Axis** property group.

#### yAxisLabel

Specify label to display to the left of the y-axis.

This property is in the **Y-Axis** property group.

#### yAxisLabelTextHeight

Specify the height of the y-axis labels in pixels.

This property is in the **Y-Axis** property group.

#### yAxisMajorDivisions

Specify the number of major divisions on the y-axis. Major divisions are separated by horizontal grid lines. See "gridColor" on page 485.

This property is in the **Y-Axis** property group.

#### yAxisMinLabelWidth

Specify the minimum width of the y-axis labels in pixels.

This property is in the **Y-Axis** property group.

#### yAxisMinorDivisions

Specify the number of minor divisions on the y-axis.

This property is in the **Y-Axis** property group.

#### yAxisMultiRangeFlag

Select to have one axis per trace, with each trace having its own range. The first trace is drawn on the outer left of the graph. The remaining traces are drawn on the inner left of the trace area.

Otherwise, all traces are plotted against a single y-axis.

The default is enabled.

This property is in the **Y-Axis** property group.

#### yAxisPercentFlag

Select to show the percent changed from the first data point instead of values for the y-axis.

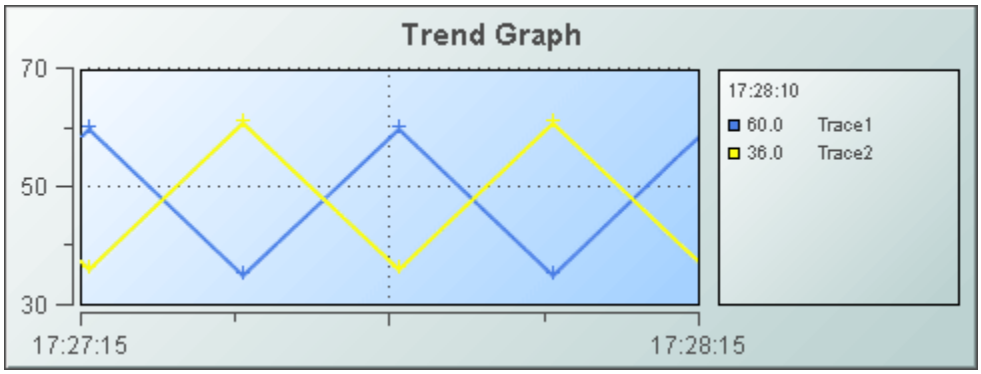
This property is in the **Y-Axis** property group.

## Trend graphs

Trend graphs visualize live and historical, time-indexed, quantitative data. Each graph contains one or more traces, and each trace visualizes tabular data that includes one, two, or three columns:

- One-column data contains a numerical column. The dashboard assigns a time stamp to each row as the data is received.
- Two-column data contains a time-valued column and a numerical column.

- Three-column data contains a time-valued column, a numerical column, and a string (data-label) column.




Use the traceCount property to specify the number of traces to be included in the chart. Use the traceNValue and traceNValueTable properties to attach data to the $N^{th}$ overlay.

Alternatively, enable multiTraceTableFlag and use multiTraceHistoryValueTable and multiTraceCurrentValueTable in order to specify data for multiple traces by using a single history attachment and a single current value attachment.

Trend graphs include the following visualization objects from the **Trends** tab:

- **Stock Chart**
- **Sparkline**
- **Single Variable Trend**
- **Multiple Variable Trend**
- **Filled Trend**
- **Threshold Trend**
- **Trend with Banded Group**

Trend graphs include the following visualization objects from the **Trends HTML5** tab:

- **Sparkline**
- **Single Variable Trend**
- **Multiple Variable Trend**
- **Filled Trend**
- **Threshold Trend**
- **Trend with Trace Group**

These visualizations all share the same properties. They differ from one another only with regard to their default values for these properties. When any of these objects is selected in the Builder canvas, the **Object Class Name** that appears at the top of the **Object Properties** pane is **obj_stockchart**.

The **Object Properties** panel organizes trend graph properties into the groups below.

## Trend graph: Alert group

Properties in this group allow you to specify changes in the appearance of trace lines, and trace markers in response to changes in the status of plotted data elements. See also traceNValueAlarmStatus and traceNValueAlarmStatusTable in the Trend graph: TraceN group property group.

### Alert group properties

This group includes the following properties:


- "valueHighAlarm" on page 500
- "valueHighAlarmEnabledFlag" on page 500
- "valueHighAlarmLineVisFlag" on page 500
- "valueHighAlarmMarkColor" on page 500
- "valueHighAlarmMarkStyle" on page 500
- "valueHighAlarmTraceColor" on page 500
- "valueHighAlarmTraceStyle" on page 501
- "valueHighWarning" on page 501
- "valueHighWarningEnabledFlag" on page 501
- "valueHighWarningLineVisFlag" on page 501
- "valueHighWarningMarkColor" on page 501
- "valueHighWarningMarkStyle" on page 501
- "valueHighWarningTraceColor" on page 501
- "valueHighWarningTraceStyle" on page 502
- "valueLowAlarm" on page 502
- "valueLowAlarmEnabledFlag" on page 502
- "valueLowAlarmLineVisFlag" on page 502
- "valueLowAlarmMarkColor" on page 502
- "valueLowAlarmMarkStyle" on page 502
- "valueLowAlarmTraceColor" on page 503
- "valueLowAlarmTraceStyle" on page 503
- "valueLowWarning" on page 503
- "valueLowWarningEnabledFlag" on page 503

- "valueLowWarningLineVisFlag" on page 503
- "valueLowWarningMarkColor" on page 503
- "valueLowWarningMarkStyle" on page 503
- "valueLowWarningTraceColor" on page 504
- "valueLowWarningTraceStyle" on page 504


#### valueHighAlarm

Specifies the threshold value used by valueHighAlarmLineVisFlag, valueHighAlarmMarkColor, valueHighAlarmMarkStyle, valueHighAlarmTraceColor, and valueHighAlarmTraceStyle.

This property is in the **Alert** property group.

#### valueHighAlarmEnabledFlag

Select to enable the high alarm threshold. See valueHighAlarm.

This property is in the **Alert** property group.

#### valueHighAlarmLineVisFlag

Select to display a dotted line at the high alarm threshold. The color of the line is set to valueHighAlarmMarkColor. This line is displayed only if valueHighAlarmEnabledFlag is selected.

This property is in the **Alert** property group.

#### valueHighAlarmMarkColor

When a trace marker's value is greater than or equal to valueHighAlarm, the marker changes to `valueHighAlarmMarkColor` and valueHighAlarmMarkStyle, provided valueHighAlarmEnabledFlag is selected. But see also traceNValueAlarmStatus and traceNValueAlarmStatusTable.

This property is in the **Alert** property group.

#### valueHighAlarmMarkStyle

When a trace marker's value is greater than or equal to valueHighAlarm, the marker changes to valueHighAlarmMarkColor and `valueHighAlarmMarkStyle`, provided valueHighAlarmEnabledFlag is selected. But see also traceNValueAlarmStatus and traceNValueAlarmStatusTable.

This property is in the **Alert** property group.

#### valueHighAlarmTraceColor

When the value of any segment of a trace line is greater than or equal to valueHighAlarm, that segment of the trace line changes to `valueHighAlarmTraceColor` and valueHighAlarmTraceStyle, provided valueHighAlarmEnabledFlag is selected.

This property is in the **Alert** property group.

#### valueHighAlarmTraceStyle

When the value of any segment of a trace line is greater than or equal to valueHighAlarm, that segment of the trace line changes to valueHighAlarmTraceStyle and `valueHighAlarmTraceStyle`, provided valueHighAlarmEnabledFlag is selected.

This property is in the **Alert** property group.

#### valueHighWarning

Specifies the threshold value used by valueHighWarningLineVisFlag, valueHighWarningMarkColor, valueHighWarningMarkStyle, valueHighWarningTraceColor, and valueHighWarningTraceStyle.

This property is in the **Alert** property group.

#### valueHighWarningEnabledFlag

Select to enable the high warning threshold. See valueHighWarning.

This property is in the **Alert** property group.

#### valueHighWarningLineVisFlag

Select to display a dotted line at the high warning threshold. The color of the line is set to valueHighWarningMarkColor. This line is displayed only if valueHighWarningEnabledFlag is selected.

This property is in the **Alert** property group.

#### valueHighWarningMarkColor

When a trace marker's value is greater than or equal to valueHighWarning but less than valueHighAlarm, the marker changes to `valueHighWarningMarkColor` and valueHighWarningMarkStyle, provided valueHighWarningEnabledFlag is selected. But see also traceNValueAlarmStatus and traceNValueAlarmStatusTable.

This property is in the **Alert** property group.

#### valueHighWarningMarkStyle

When a trace marker's value is greater than or equal to valueHighWarning but less than valueHighAlarm, the marker changes to valueHighWarningMarkColor and `valueHighWarningMarkStyle`, provided valueHighWarningEnabledFlag is selected. But see also traceNValueAlarmStatus and traceNValueAlarmStatusTable.

This property is in the **Alert** property group.

#### valueHighWarningTraceColor

When the value of any segment of a trace line is greater than or equal to valueHighWarning property but less than valueHighAlarm, that segment of the trace

line changes to `valueHighWarningTraceColor` and valueHighWarningTraceStyle, provided valueHighWarningEnabledFlag is selected.

This property is in the **Alert** property group.

#### valueHighWarningTraceStyle

When the value of any segment of a trace line is greater than or equal to valueHighWarning property but less than valueHighAlarm, that segment of the trace line changes to valueHighWarningTraceColor and `valueHighWarningTraceStyle`, provided valueHighWarningEnabledFlag is selected.

This property is in the **Alert** property group.

#### valueLowAlarm

Specifies the threshold value used by valueLowAlarmLineVisFlag, valueLowAlarmMarkColor, valueLowAlarmMarkStyle, valueLowAlarmTraceColor, and valueLowAlarmTraceStyle.

This property is in the **Alert** property group.

#### valueLowAlarmEnabledFlag

Select to enable the low alarm threshold. See valueLowAlarm.

This property is in the **Alert** property group.

#### valueLowAlarmLineVisFlag

Select to display a dotted line at the low alarm threshold. The color of the line is set to valueLowAlarmMarkColor. This line is displayed only if valueLowAlarmEnabledFlag is selected.

This property is in the **Alert** property group.

#### valueLowAlarmMarkColor

When a trace marker's value is less than or equal to valueLowAlarm, the marker changes to `valueLowAlarmMarkColor` and valueLowAlarmMarkStyle, provided valueLowAlarmEnabledFlag is selected. But see also traceNValueAlarmStatus and traceNValueAlarmStatusTable.

This property is in the **Alert** property group.

#### valueLowAlarmMarkStyle

When a trace marker's value is less than or equal to valueLowAlarm, the marker changes to valueLowAlarmMarkColor and `valueLowAlarmMarkStyle`, provided valueLowAlarmEnabledFlag is selected. But see also traceNValueAlarmStatus and traceNValueAlarmStatusTable.

This property is in the **Alert** property group.

#### valueLowAlarmTraceColor

When the value of any segment of a trace line is less than or equal to valueLowAlarm, that segment of the trace line changes to `valueLowAlarmTraceColor` and valueLowAlarmTraceStyle, provided valueLowAlarmEnabledFlag is selected.

This property is in the **Alert** property group.

#### valueLowAlarmTraceStyle

When the value of any segment of a trace line is less than or equal to valueLowAlarm, that segment of the trace line changes to valueLowAlarmTraceColor and `valueLowAlarmTraceStyle`, provided valueLowAlarmEnabledFlag is selected.

This property is in the **Alert** property group.

#### valueLowWarning

Specifies the threshold value used by valueLowWarningLineVisFlag, valueLowWarningMarkColor, valueLowWarningMarkStyle, valueLowWarningTraceColor, and valueLowWarningTraceStyle.

This property is in the **Alert** property group.

#### valueLowWarningEnabledFlag

Select to enable the low warning threshold. See valueLowWarning.

This property is in the **Alert** property group.

#### valueLowWarningLineVisFlag

Select to display a dotted line at the low warning threshold. The color of the line is set to valueLowWarningMarkColor. This line is displayed only if valueLowWarningEnabledFlag is selected.

This property is in the **Alert** property group.

#### valueLowWarningMarkColor

When a trace marker's value is less than or equal to valueLowWarning but greater than valueLowAlarm, the marker changes to `valueLowWarningMarkColor` and valueLowWarningMarkStyle, provided valueLowWarningEnabledFlag is selected. But see also traceNValueAlarmStatus and traceNValueAlarmStatusTable.

This property is in the **Alert** property group.

#### valueLowWarningMarkStyle

When a trace marker's value is less than or equal to valueLowWarning but greater than valueLowAlarm, the marker changes to valueLowWarningMarkColor and `valueLowWarningMarkStyle`, provided valueLowWarningEnabledFlag is selected. But see also traceNValueAlarmStatus and traceNValueAlarmStatusTable.

This property is in the **Alert** property group.

#### valueLowWarningTraceColor

When the value of any segment of a trace line is less than or equal to valueLowWarning but greater than valueLowAlarm, that segment of the trace line changes to `valueLowWarningTraceColor` and valueLowWarningTraceStyle, provided valueLowWarningEnabledFlag is selected.

This property is in the **Alert** property group.

#### valueLowWarningTraceStyle

When the value of any segment of a trace line is less than or equal to valueLowWarning property but greater than valueLowAlarm, that segment of the trace line changes to valueLowWarningTraceColor and `valueLowWarningTraceStyle`, provided valueLowWarningEnabledFlag is selected.

This property is in the **Alert** property group.

## Trend graph: Background group

Properties in this group control the visibility and appearance of the portion of the graph that serves as the background of both the plot area and legend.

#### Background group properties

The group contains the following properties:

- "bgBorderFlag" on page 504
- "bgColor" on page 505
- "bgEdgeWidth" on page 505
- "bgGradientColor2" on page 505
- "bgGradientMode" on page 505
- "bgRaisedFlag" on page 505
- "bgRoundness" on page 506
- "bgShadowFlag" on page 506
- "bgStyleFlag" on page 506
- "bgVisFlag" on page 506
- "borderPixels" on page 506

#### bgBorderFlag

Select to display a border around the background rectangle.

This property is in the **Background** property group.

#### bgColor

Sets the background color. Select the **...** button and choose a color from the palette. Close the Color Chooser window when you are done.

This property is in the **Background** property group.

#### bgEdgeWidth

Sets the width in pixels of the 3D edge on the background rectangle. This property is only used if `bgBorderFlag` is selected.

This property is in the **Background** property group.

#### bgGradientColor2

Sets the color for the second color in the gradient. The default is white. The `bgColor` property sets the first color in the gradient.

This property is in the **Background** property group.

#### bgGradientMode

Display a gradient in the background rectangle. Select from the following options:

- **None**: No gradient.
- **Diagonal Edge**: Gradient is drawn at a 45 degree angle from the top left to the bottom right corner of the object.
- **Diagonal Center**: Gradient is drawn at a 45 degree angle from the center to the top left and the bottom right corners of the object.
- **Horizontal Edge**: Gradient is drawn horizontally from the top to the bottom of the object.
- **Horizontal Center**: Gradient is drawn horizontally from the center to the top and bottom of the object.
- **Vertical Edge**: Gradient is drawn vertically from the left to the right of the object.
- **Vertical Center**: Gradient is drawn vertically from the center to the left and right of the object.

This property is in the **Background** property group.

#### bgRaisedFlag

Reverses the direction of the gradient, as well as that of the 3D edge if the `bgStyle` selected is 3D Rectangle.

This property is in the **Background** property group.

#### bgRoundness

Sets the arc length of the rounded corners. This property is only available if the `bgStyle` selected is **Round Rectangle**.

The value of `bgRoundness` cannot exceed half the value of the `objWidth` or the `objHeight`. If `bgRoundness` does exceed that value, half of `objWidth` or `objHeight` (whichever is smaller) will be used instead. For example if `objWidth` is `100` and `objHeight` is `50`, then the value of `bgRoundness` cannot exceed `25`. If it does, then half the value of `objHeight` (`25`) will be used instead.

This property is in the **Background** property group.

#### bgShadowFlag

Select to display a drop shadow on the background rectangle.

This property is in the **Background** property group.

#### bgStyleFlag

Choose one of the following three options from the drop down menu:

- **Rectangle**: Select to display a background rectangle.
- **3D Rectangle**: Select to display a 3D edge on the background rectangle. If selected, use `bgEdgeWidth` to set the width of the 3D edge.
- **Round Rectangle**: Select to display a background rectangle with rounded edges. If selected, use `bgRoundness` to set the arc length of the rounded corners.

This property is in the **Background** property group.

#### bgVisFlag

Select to display the background rectangle.

This property is in the **Background** property group.

#### borderPixels

Sets the width in pixels of the border between the chart and the edge of the background rectangle.

This property is in the **Background** property group.

## Trend graph: Data group

Properties in this group control the y-axis range, as well as the maximum number of data points contained in the chart. It also contains a flag that controls whether only historical data is included.

#### Data group properties

The group contains the following properties:


- "historyOnlyFlag" on page 507
- "maxPointsPerTrace" on page 507
- "yValueMax" on page 507
- "yValueMin" on page 507


#### historyOnlyFlag

When checked, the graph plots only data that is applied to the `traceNValueTable` properties and will ignore the `timeShift` property and any data that is applied to the `traceNValue` properties. This is useful when the same graph instance is to be used to view either historical data or historical data together with current data by setting substitutions on the display.

The default is unchecked.

This property is in the **Data** property group.

#### maxPointsPerTrace

The maximum number of data points contained in the chart. Specify a value between 2 and 30000, inclusive.

The default is 1000.

This property is in the **Data** property group.

#### yValueMax

Determines the range of the y-axis if the yAxisAutoScaleMode is set to **Off**. Select **On** for the yAxisAutoScaleMode to calculate the y-axis range according to data values being plotted. To calculate a y-axis range that always includes yValueMin and `yValueMax`, select **On - Include Min/Max**.

This property is used only if yAxisMultiRangeMode is set to **Off** or **Classic**.

This property is in the **Data** property group.

#### yValueMin

Controls the range of y-axis if the yAxisAutoScaleMode is set to **Off**. Select **On** for the yAxisAutoScaleMode to calculate the y-axis range according to data values being plotted. To calculate a y-axis range that always includes `yValueMin` and yValueMax, select **On - Include Min/Max**.

This property is used only if yAxisMultiRangeMode is set to **Off** or **Classic**.

This property is in the **Data** property group.

## Trend graph: Data Format group

This group contains the yValueFormat property, which controls the format of displayed values.

#### yValueFormat

Select or enter the numeric format of values displayed in the legend and popup legend. To enter a format, use syntax from the Java `DecimalFormat` class.

This property is in the **Data Format** property group.

## Trend graph: Interaction group

Properties in this group control various forms of interaction between the end user and the chart, including scrolling and activating commands, drill downs, and tooltips.

#### Interaction group properties

This group contains the following properties:


- "commandCloseWindowOnSuccess" on page 508
- "command" on page 509
- "commandConfirm" on page 509
- "confirmText" on page 510
- "cursorColor" on page 510
- "cursorFlag" on page 510
- "drillDownSelectMode" on page 510
- "drillDownTarget" on page 510
- "scrollbarMode" on page 511
- "scrollbarSize" on page 512
- "zoomEnabledFlag" on page 512


#### commandCloseWindowOnSuccess

Select this property to automatically close the window that initiates a **SYSTEM** command when the command is executed successfully. This applies to **SYSTEM** commands only, and is not supported at all for thin-client, Web-page deployments.

With **APAMA** commands, the window is closed whether or not the command is executed successfully. For **MULTIPLE** commands, the window closes when the first command in the command group succeeds.

This property is in the **Interaction** property group.

#### command

Assign a command or group of commands to this stock chart by right-clicking on the `command` property name in the Object Properties window. Select **Define Command** and choose **SYSTEM**, **APAMA**, or **MULTIPLE**. See "Using the Define Apama Command dialog" on page 226.

Once a command or command group has been assigned to this object, you can activate it from a deployed dashboard or from the Dashboard Builder:

- Dashboard Builder: Double click on the object.
- Web-based deployment: Single click on the object or else right click on it and select **Execute Command** from the popup menu.
- Local deployment: By default, single-click on the object or else right-click on it and select **Execute Command** from the popup menu. To override the default, select **Tools > Options** in the Builder (do this before you generate the deployment package), and uncheck **Single-Click for Drill Down and Commands** in the **General** tab. This allows the end user to use either a double click or a right click.

When you activate a command, any defined drill down substitutions are performed, and then the command is executed.

If you assign multiple commands, the commands are launched in an arbitrary order, and are executed asynchronously; there is no guarantee that one command will finish before the next one in the sequence starts.

This property is in the **Interaction** property group.

#### commandConfirm

By default, when the end user executes a command (see command), the command confirmation dialog is disabled. To control this option for each individual object, use the `commandConfirm` check box. If confirmation is required for a **MULTIPLE** command group, a single confirmation dialog is presented; if you confirm the execution, all individual commands in the group are executed with no further confirmation. If the you cancel the execution, none of the commands in the group is executed.

You can also override the confirmation status of individual objects with an application-wide policy. Select **Tools > Options** and choose from three confirmation values:

- **Do not confirm**: Indicates that no commands require confirmation (regardless of each object's confirmation status).
- **Confirm all**: Indicates that all commands require confirmation (regardless of each object's confirmation status).
- **Use object confirm flag** (default): Indicates that the confirmation status of each object will determine whether confirmation is required.

This property is in the **Interaction** property group.

#### confirmText

Use this property to write your own text for the confirmation dialog. Otherwise, default text is used. See commandConfirm.

This property is in the **Interaction** property group.

#### cursorColor

Sets the color of the cursor, as well as the zoom-area rectangle (see zoomEnabledFlag). Select the **...** button and choose a color from the palette. Close the Color Chooser window when you are done.

The default is grey.

This property is in the **Interaction** property group.

#### cursorFlag

Select to enable the cursor. When the cursor is enabled, select the chart and point to a location on a trace to see a cursor line at that location and display the time and values of all traces at the cursor line on the legend. Select the legendPopupFlag to display the legend along the cursor.

The cursor is enabled by default.

This property is in the **Interaction** property group.

#### drillDownSelectMode

Use this property to control how a drill down display is activated. Select one of the following:

- **Anywhere** to activate a drill down display by double-clicking anywhere on the chart.
- **Element Only** to enable a drill down display only when you double-click on an element of the chart, that is, a trace point.

This property is in the **Interaction** property group.

#### drillDownTarget

To specify a drill down display, double click on `drillDownTarget` in the **Property Name** field to bring up the Drill Down Properties dialog. See "Drill-Down Specification" on page 535.

Trend graphs support drill down from a trace point. If the trend graph has a `drillDownTarget` specified, clicking on a trace point sets the following predefined substitutions:

- $traceNumber: number of the trace (1 to 10) that contains the selected point
- $traceLabel: label of selected trace
- $pointValue: y value of point

- $pointTimestamp: timestamp of point
- $pointLabel: data label (if any) of point
- $pointIndex: position of point in trace data (0 to maxPointsPerTrace)

If the `drillDownSelectMode` property is set to **Element Only**, clicks on the graph that are not near a trace point are ignored. If `drillDownSelectMode` is set to Anywhere, a click anywhere on the graph triggers a drill down, but if the click is not near a trace point the substitutions listed above are not set.

Thin client (Display Server) deployments support mouseover text and drill down from data points on traces on the trend graph. If the trend graph's `cursorFlag` property is checked, this enables mouseover on the trace points. If the mouse is over a trace point, a browser tooltip box appears displaying the legend values that correspond to that point. Following are the limitations on this feature:

- If this feature is used on a graph with many trace points, the performance of the browser may be sluggish when the display is loading or refreshing. To avoid this, set the `timeRange` property so that only a portion of the trace points are visible at a time.
- If a thin client display refresh occurs while positioning the mouse over a point, the browser tooltip may not appear or it may appear in the wrong location.
- When `maxPointsPerTrace` is exceeded on a trace (1000 by default), an old trace point is shifted out of the trace for each new point that is added. If this occurs between the time that the thin client display was last refreshed and the time that the user clicks on a point, the drilldown substitutions reflect the new set of data points. For example, if two points are shifted out of the trace, the drilldown substitutions are set as though the selected point were two positions to the right of the point the user actually clicked.

This property is in the **Interaction** property group.

#### scrollbarMode

Sets whether and when the scroll bar appears in the chart. Select one of the following from the drop down menu:

- **Never**: Default setting
- **Always**: Display a scroll bar at all times.
- **As Needed**: Display the scroll bar when necessitated by zooming in the trace area, or when more data is loaded into the chart than is displayed in the time range. For example, if the time range of the data in your data attachment is greater than timeRange, setting scrollbarMode to **As Needed** will enable a scroll bar, allowing the end user to view all data loaded into the chart.

This property is in the **Interaction** property group.

#### scrollbarSize

Specify the height of the horizontal scroll bar and the width of the vertical scroll bar, in pixels. The default value is -1, which sets the size to the system default.

This property is in the **Interaction** property group.

#### zoomEnabledFlag

Select to enable zooming within the chart. Click in the chart's trace area and drag the cursorColorcursor until a desired range is selected. While dragging, a rectangle is drawn to show the zoom area. The rectangle's default color is yellow (this can be changed in the property). After the zoom is performed, the chart stores up to four zoom operations in queue. To zoom out, press the shift key and click in the chart's trace area.

This property is in the **Interaction** property group.

## Trend graph: Label group

Properties in this group control the chart's main label (which defaults to **Stock Chart**), including text, alignment, color, font, and size.

#### Label group properties

The group includes the following properties:

- "label" on page 512
- "labelTextAlignX" on page 512
- "labelTextColor" on page 513
- "labelTextFont" on page 513
- "labelTextHeight" on page 513

#### label

Specifies the text for the chart label. Click the ellipsis for multi-line text.

The default is **Single Variable Trend**, **Multiple Variable Trend**, **Filled Trend**, **Threshold Trend**, or **Single Trend with Marks**.

This property is in the **Label** property group.

#### labelTextAlignX

Sets the alignment of the chart label (see the label property). Select **Left**, **Center**, or **Right** from the drop down list.

This property is in the **Label** property group.

#### labelTextColor

Specifies the color of the chart label text (see the label property). Select the **...** button and choose a color from the palette. Close the Color Chooser window when you are done.

This property is in the **Label** property group.

#### labelTextFont

Specifies the font of the chart label text (see the label property). Select an item from drop down list.

This property is in the **Label** property group.

#### labelTextHeight

Specifies the point size of the chart label text (see the label property).

This property is in the **Label** property group.

## Trend graph: Legend group

Properties in this group control the visibility, appearance, and content of the chart legend.

#### Legend group properties

The group contains the following properties:


- "legendBgColor" on page 513
- "legendBgGradientFlag" on page 513
- "legendPopupFlag" on page 514
- "legendTimeFormat" on page 514
- "legendValueMinSpace" on page 514
- "legendVisFlag" on page 514
- "legendWidthPercent" on page 514


#### legendBgColor

Select the **...** button and choose a color from the palette to set the background color of the legend. Close the Color Chooser window when you are done.

This property is in the **Legend** property group.

#### legendBgGradientFlag

Select the `legendBgGradientFlag` to display a gradient in the legend background.

This property is in the **Legend** property group.

#### legendPopupFlag

When the cursorFlag property is enabled, select `legendPopupFlag` to display the legend along the cursor.

This property is in the **Legend** property group.

#### legendTimeFormat

Sets the format for the time displayed in the legend. Use syntax from the Java `SimpleDateFormat` class. For example, **MMMM dd, yyyy hh:mm:ss** results in the form **August 30, 2003 05:32:12 PM**. If no format is given, the timeFormat is used.

This property is in the **Legend** property group.

#### legendValueMinSpace

Specify the minimum distance in pixels between values and labels in the legend.

This property is in the **Legend** property group.

#### legendVisFlag

Select to display the legend.

This property is in the **Legend** property group.

#### legendWidthPercent

Sets the percent of the total width of the object used for the legend.

This property is in the **Legend** property group.

## Trend graph: Marker group

Properties in this group control the size of trace markers.

#### Marker group properties

The group includes the following properties:

- "markDefaultSize" on page 514
- "markScaleMode" on page 514

#### markDefaultSize

Sets the size of the markers (see `traceNMarkStyle`) in pixels. Supply an integer value that is between 1 and 18, inclusive.

This property is in the **Marker** property group.

#### markScaleMode

Select one of the following from the drop down menu to set the scale mode:

- No Scale: All makers, across and within traces, are the same size.
- Scale by Trace: Scale markers according to the trace in which they reside, that is, markers in the first trace are the largest, across all traces, and the markers in the last trace are the smallest.
- Scale Within Trace: Scale markers according to the relative temporal order of the data within each trace, that is, the marker for the earliest data in any given trace is the smallest in that trace and the marker for the latest data in the trace is the largest in that trace.

This property is in the **Marker** property group.

## Trend graph: Object group

Properties in this group control the visibility and transparency of the chart as a whole. They also control (or reflect) the overall position and dimensions of the chart. In addition, a property in this group reflects the generated name of this individual chart.

### Object group properties

This group contains the following properties:


- "anchor" on page 515
- "dock" on page 515
- "objHeight" on page 516
- "objName" on page 516
- "objWidth" on page 516
- "objX" on page 516
- "objY" on page 516
- "transparencyPercent" on page 516
- "visFlag" on page 516


#### anchor

Select zero or more of **Top**, **Left**, **Bottom**, and **Right** in order to control the object's placement. The `anchor` property is only applied when the display is resized either by changing the **Background Properties** on the display or by resizing the window in **Layout** mode. If an object has the `dock` property set, the `anchor` property is ignored. See "About resize modes" on page 39.

#### dock

Select **None** (default), **Top**, **Left**, **Bottom**, **Right**, or **Fill** in order to control the object's placement in **Layout** resize mode. See "About resize modes" on page 39.

#### objHeight

Sets the height of a chart by entering a value for this property or by dragging a handle of the bounding box that appears when the stock chart is selected. When you drag a handle of the bounding box, the displayed value for this property changes to reflect the real-time height of the chart.

This property is in the **Object** property group.

#### objName

An identifier that is generated by the Dashboard Builder. This name can be used by other objects' properties in order to refer to the named stock chart.

This property is in the **Object** property group.

#### objWidth

Sets the width of a chart by entering a value for this property or by dragging a handle of the bounding box that appears when the stock chart is selected. When you drag a handle of the bounding box, the displayed value for this property changes to reflect the real-time width of the chart.

This property is in the **Object** property group.

#### objX

Sets the X coordinate of the center of this visualization object, relative to the lower left corner of the current dashboard. This value is set automatically when you position the object with the mouse.

This property is in the **Object** property group.

#### objY

Sets the Y coordinate of the center of this visualization object, relative to the lower left corner of the current dashboard. This value is set automatically when you position the object with the mouse.

This property is in the **Object** property group.

#### transparencyPercent

Sets the transparency of this chart.

This property is in the **Object** property group.

#### visFlag

Deselect to make this visualization object invisible in the current dashboard.

This property is in the **Object** property group.

## Trend graph: Plot Area group

Properties in this group control the appearance of the plot area, the rectangular area that serves as background for the traces (but not for the legend or axis labels; see "Trend graph: Background group" on page 504). There is also a property that controls the color of the horizontal grid line or lines.

### Plot Area group

This group contains the following properties:

- "gridColor" on page 517
- "traceBgColor" on page 517
- "traceBgGradientFlag" on page 517
- "traceBgImage" on page 517

### gridColor

Sets the color of the dotted, horizontal midline of the plot area. Select the **...** button and choose a color from the palette. Close the Color Chooser window when you are done.

This property is in the **Plot Area** property group.

### traceBgColor

To set the color of the plot area, select the **...** button and choose a color from the palette to set the background color. Close the Color Chooser window when you are done.

This property is in the **Plot Area** property group.

### traceBgGradientFlag

Select to display a gradient in the plot area.

This property is in the **Plot Area** property group.

### traceBgImage

Specify an image (`.gif`, `.jpg`, or `.png` file) to display in the plot area. Select the name of the image file from the drop down menu, or enter the pathname of the file. The drop down menu contains the names of image files located in the current directory (by default, the `dashboards` directory of your Apama installation's work directory), as well as image files located in the first level of subdirectories. If you enter a pathname, use an absolute pathname or a pathname that is relative to the current directory.

This property is in the **Plot Area** property group.

## Trend graph: Trace group

Properties in this group control the number of traces that the chart contains, as well as the trace fill style. They can also specify the trace data, if a single data table is used for multiple traces.

### Trace group properties

The group contains the following properties:


- "multiTraceCurrentValueTable" on page 518
- "multiTraceHistoryValueTable" on page 518
- "multiTraceTableFlag" on page 519
- "traceCount" on page 519
- "traceFillStyle" on page 520


#### multiTraceCurrentValueTable

To display current data for multiple traces by using a single attachment, attach a table to this property. The first column in the data table must be a timestamp column. The remaining columns are expected to be Y data values to be plotted. The *N*th data column is used for trace *N*'s data, and the column name is used for `traceNLabel` (if not already assigned).

If the `multiTraceTableFlag` is checked, the number of traces whose properties are shown in the Builder's property sheet is determined by the number of data columns in the data table attachments or by the `traceCount` property, whichever is larger. However, the number of traces that are plotted on the graph is determined by the number of data columns in the data table attachments.

Typically, the data attachment for `multiTraceHistoryValueTable` provides the initial data points to be plotted while `multiTraceCurrentValueTable` provides the new data points to be plotted while the display is viewed. If a trace plots only historical or only current data, only one of the properties needs to be attached to data. However if both properties are attached to data, be sure that the tables applied to both have the same number and type of columns.

When `multiTraceTableFlag` is checked, the properties `traceNValueTable` and `traceNValue` (for *N* between 1 and 10, inclusive) are not shown in the property sheet, since all trace data is expected to be provided via `multiTraceCurrentValueTable` or `multiTraceHistoryValueTable`.

This property is in the **Trace** property group.

#### multiTraceHistoryValueTable

To display historical data for multiple traces by using a single attachment, attach a table to this property. The first column in the data table must be a timestamp column. The remaining columns are expected to be Y data values to be plotted. The *N*th data column

is used for trace *N*'s data, and the column name is used for `traceNLabel` (if not already assigned).

If `multiTraceTableFlag` is checked, the number of traces whose properties are shown in the Builder's property sheet is determined by the number of data columns in the data table attachments or by the `traceCount` property, whichever is larger. However, the number of traces that are plotted on the graph is determined by the number of data columns in the data table attachments.

Typically, the data attachment for `multiTraceHistoryValueTable` provides the initial data points to be plotted while `multiTraceCurrentValueTable` provides the new data points to be plotted while the display is viewed. If a trace plots only historical or only current data, only one of the properties needs to be attached to data. However if both properties are attached to data, be sure that the tables applied to both have the same number and type of columns.

When `multiTraceTableFlag` is checked, the properties `traceNValueTable` and `traceNValue` (for *N* between 1 and 10, inclusive) are not shown in the property sheet, since all trace data is expected to be provided via `multiTraceCurrentValueTable` or `multiTraceHistoryValueTable`.

This property is in the **Trace** property group.

#### multiTraceTableFlag

Controls whether data for multiple traces can be attached to the graph with a single table. When checked, the properties `multiTraceCurrentValueTable` and `multiTraceHistoryValueTable`, are shown in the Trace category.

When `multiTraceTableFlag` is checked, the properties `traceNValueTable` and `traceNValue` (for *N* between 1 and 10, inclusive) are not shown in the property sheet, since all trace data is expected to be provided via `multiTraceCurrentValueTable` or `multiTraceHistoryValueTable`.

If `multiTraceTableFlag` is checked, the number of traces whose properties are shown in the Builder's property sheet is determined by the number of data columns in the data table attachments or by the `traceCount` property, whichever is larger. However, the number of traces that are plotted on the graph is determined by the number of data columns in the data table attachments.

This property is in the **Trace** property group.

#### traceCount

Sets the number of traces. The maximum is ten. For each overlay, the Dashboard Builder automatically creates a set of properties in the Object Properties window.

This property is in the **Trace** property group.

#### traceFillStyle

Specifies the effect with which to fill the area from the trace line to the bottom of the graph. The color is determined by traceNLineColor. Select one of the following fill styles from the drop down menu:

- **Solid**
- **Transparent**
- **Gradient**
- **Transparent Gradient**
- **None** (default)

This property is in the **Trace** property group.

## Trend graph: Trace*N* group

The properties in this group control the visibility and appearance of the price trace, as well as the data to which it is attached (unless multiTraceTableFlag is enabled). They also control y-axis visibility, scaling, and labeling. In addition, there are properties to which you can attach a data table that indicates the alarm status of plotted data; see "Trend graph: Alert group" on page 499.

#### Trace*N* group properties

The group includes the following properties:


- "traceNLabel" on page 521
- "traceNLineColor" on page 521
- "traceNLineStyle" on page 521
- "traceNLineThickness" on page 521
- "traceNMarkColor" on page 522
- "traceNMarkStyle" on page 522
- "traceNType" on page 522
- "traceNValue" on page 523
- "traceNValueAlarmStatus" on page 523
- "traceNValueAlarmStatusTable" on page 524
- "traceNValueDivisor" on page 524
- "traceNValueHistoryFlag" on page 525
- "traceNValueTable" on page 525
- "traceNVisFlag" on page 525

- "traceNYAxisAutoScaleMode" on page 525
- "traceNYAxisFlag" on page 525
- "traceNYAxisGridVisFlag" on page 526
- "traceNYAxisMinLabelWidth" on page 526
- "traceNYAxisValueLabels" on page 526
- "traceNYAxisValueMax" on page 526
- "traceNYAxisValueMin" on page 526


**trace*N*Label**

Enter a label for the trace line. This label appears in the chart's legend.

This property is in the **Trace***N* property group.

**trace*N*LineColor**

Sets the trace line color. Select the **...** button and choose a color from the palette. Close the Color Chooser window when you are done.

This property is in the **Trace***N* property group.

**trace*N*LineStyle**

Select one of the following line styles for the trace line from the drop down menu:

- **No Line**
- **Solid**
- **Dotted**
- **Dashed**
- **Dot Dashed**

This property is in the **Trace***N* property group.

**trace*N*LineThickness**

Select one of the following thickness specifications for the price trace line from the drop down menu:

- **Thin**
- **Medium**
- **Thick**

This property is in the **Trace***N* property group.

#### traceNMarkColor

Select the **...** button and choose a color from the palette to set the trace marker color. Close the Color Chooser window when you are done.

This property is in the **Trace***N* property group.

#### traceNMarkStyle

Sets the style of the marker used on the trace. Select one of the following items from the drop down menu:

- No Marker
- Dot
- +
- *
- o
- x
- Filled Circle
- Filled Diamond
- Filled Triangle
- Filled Square
- Filled Star

This property is in the **Trace***N* property group.

#### traceNType

Sets the trace type. The valid values are **Line** (the default), **Bar**, and **Event**.

A **Bar** type trace draws a vertical bar for each data point, from zero to the point's Y value. The bar is just a vertical line whose width is determined by `traceNLineThickness`. The `traceNLineColor` and `traceNLineStyle` also control the appearance of the bar. If the point exceeds an alarm limit specified on the graph, the alarm color is used for the bar color. If `traceNMarkStyle` is set to any value other than **None**, the mark is drawn at the end of the bar.

For an **Event** type trace, no line is drawn. Instead, a small rectangle containing a single text character is drawn for each data point. The character is the first character of the corresponding data label if any, otherwise it is the first character of the trace label. The `traceNColor` property determines the color of edges of the box and the text character, unless the point exceeds an alarm limit specified on the graph, in which case the corresponding alarm color is used. The box's fill color is set to `traceNMarkColor` or the appropriate alarm mark color, if any. However, if the mark color is the same as the color used for the box edge and text, `traceBgColor` is used as the box fill color instead.

Each event box is positioned vertically according to the Y data value for the corresponding data point. However, if `traceN` is attached to a data table that provides data labels but no Y data values, an Event trace is plotted regardless of the `traceNType` setting. The event boxes are all drawn near the bottom of the trace area.

This property is in the **Trace***N* property group.

#### trace*N*Value

To display current data, attach to this property. When you attach data to this property, the time displayed on the trend graph is automatically updated each time data is received. The table in your data attachment can contain either a single point of data, two columns of data, or three columns of data. If it contains a single point of data, the dashboard assigns the time stamp when the graph receives the data.

If it contains two columns of data, the first column must be the time value and the second column the value to plot.

Following are supported formats for the time value column:

- **mm/dd/yyyy hh:mm:ss** (for example, **01/16/2004 12:30:03**)
- **yyyy-mm-dd hh:mm:ss** (for example, **2004-01-16 12:30:03**)
- Number of milliseconds since midnight, January 1, 1970 UTC

In order to view all available data, you must set the properties timeRange to -1 and timeShift to a negative value. This negative value will be used to round the start and end times for the Y Axis. For example, if you specify **-15** for the `timeShift` property, the start and end times for the Y Axis will be rounded to the nearest 15 seconds.

If the attachment contains three columns of data, the third column must be a string column, which is used as the data label for the corresponding data point. The data label for a point is shown in the fixed legend and in the popup legend, between the trace value and the trace label, and is enclosed in parentheses. If the `cursorFlag` property is checked, the data label shown in the legend is for the data point that is directly under or to the left of the cursor.

This property is in the **Trace***N* property group.

#### trace*N*ValueAlarmStatus

To apply an alarm status to trace*N*, enter an alarm status index, which indicates how to determine the marker color and style for each new plotted point derived from traceNValue. Enter one of the following integers:

**0**: Normal marker color and style. See "traceNMarkColor" on page 522 and "traceNMarkStyle" on page 522.

**1**: Low alarm marker color and style: See "valueLowAlarmMarkColor" on page 502 and "valueLowAlarmMarkStyle" on page 502.

**2**: Low warning marker color and style. See "valueLowWarningMarkColor" on page 503 and "valueLowWarningMarkStyle" on page 503.

**3**: High warning marker color and style. See "valueHighWarningMarkColor" on page 501 and "valueHighWarningMarkStyle" on page 501.

**4**: High alarm marker color and style. See "valueHighAlarmMarkColor" on page 500 and "valueHighAlarmMarkStyle" on page 500.

**-1**: Determine marker color and style by comparing the value to the enabled alarm thresholds. See "valueHighAlarm" on page 500, "valueHighWarning" on page 501, "valueLowAlarm" on page 502, and "valueLowWarning" on page 503.

The default is -1.

This property is in the **Trace***N* property group.

#### trace*N*ValueAlarmStatusTable

Attach an alarm table containing status indexes to `tracenValueAlarmStatusTable` to enable rule based alarm statuses for trace markers. This table must have a time column (formatted like the time value in the traceNValueTable) and a value column where the value column contains alarm status values 0-4. The table must also have the same number of rows as the corresponding `tracenValueTable`. For each data element in traceNValueTable, the status index at the corresponding position in `tracenvalueAlarmStatusTable` is used to set the alarm status of the marker.

Valid indexes are:

**0**: Normal marker color and style. See "traceNMarkColor" on page 522 and "traceNMarkStyle" on page 522.

**1**: Low alarm marker color and style. See "valueLowAlarmMarkColor" on page 502 and "valueLowAlarmMarkStyle" on page 502.

**2**: Low warning marker color and style. See "valueLowWarningMarkColor" on page 503 and "valueLowWarningMarkStyle" on page 503.

**3**: High warning marker color and style. See "valueHighWarningMarkColor" on page 501 and "valueHighWarningMarkStyle" on page 501.

**4**: High alarm marker color and style. See "valueHighAlarmMarkColor" on page 500 and "valueHighAlarmMarkStyle" on page 500.

**-1**: Determine marker color and style by comparing the value to the enabled alarm thresholds. See "valueHighAlarm" on page 500, "valueHighWarning" on page 501, "valueLowAlarm" on page 502, and "valueLowWarning" on page 503.

If no data is attached to `tracenValueAlarmStatusTable`, then the alarm status for a trace marker is determined by comparing the marker's value to the enabled thresholds. See valueHighAlarm, valueHighWarning, valueLowAlarm, and valueLowWarning.

This property is in the **Trace***N* property group.

#### trace*N*ValueDivisor

All trace values are divided by the number entered into the `tracenValueDivisor`.

This property is in the **Trace***N* property group.

#### trace*N*ValueHistoryFlag

Do not use this property.

This property is in the **Trace***N* property group.

#### trace*N*ValueTable

To display historical data, attach to `tracenValueTable`, where n is the trace number, and include two columns in your attachment. The first column must be the time value and the second column the value to plot. Following are supported formats for the time value column:

- **mm/dd/yyyy hh:mm:ss** (for example, **01/16/2004 12:30:03**)
- **yyyy-mm-dd hh:mm:ss** (for example, **2004-01-16 12:30:03**)
- Number of milliseconds since midnight, January 1, 1970 UTC

In order to view all available data, you must set the properties timeRange to -1 and timeShift to a negative value. This negative value will be used to round the start and end times for the Y Axis. For example, if you specify **-15** for the `timeShift` property, the start and end times for the Y Axis will be rounded to the nearest 15 seconds.

This property is in the **Trace***N* property group.

#### trace*N*VisFlag

Select to control trace visibility. Mouse over a trace's entry in the legend and hold down the left mouse button in order to temporarily hide all other traces in the graph.

This property is in the **Trace***N* property group.

#### trace*N*YAxisAutoScaleMode

Controls how the y-axis range is calculated for this trace, if yAxisMultiRangeMode is set to **Multiple Axis** or **Strip Chart**. Select one of the following from the drop down menu:

- **Off**: The traceNYAxisValueMin and traceNYAxisValueMax properties determine the range of the y-axis.
- **On**: The dashboard calculates the y-axis range according to data values being plotted.
- **On - Include Min/Max**: The dashboard calculates the smallest range (with rounding) that includes traceNYAxisValueMin and traceNYAxisValueMax as well as all plotted points.

This property is in the **Trace***N* property group.

#### trace*N*YAxisFlag

Controls the visibility of the labels and ticks for trace*N*, if yAxisMultiRangeMode is set to **Multiple Axis** or **Strip Chart**.

This property is in the **Trace***N* property group.

#### trace*N*YAxisGridVisFlag

Set to display a horizontal line for each major y-axis division for trace*N*, if yAxisMultiRangeMode is set to **Multiple Axis** or **Strip Chart**.

This property is in the **Trace***N* property group.

#### trace*N*YAxisMinLabelWidth

Specifies the minimum width of the y-axis labels in pixels, if yAxisMultiRangeMode is set to **Multiple Axis** or **Strip Chart**.

This property is in the **Trace***N* property group.

#### trace*N*YAxisValueLabels

Set to display a text label or tick mark on the y-axis in place of a numerical value, if yAxisMultiRangeMode is set to **Multiple Axis** or **Strip Chart**. Include a value with no label to display a tick mark without a label. Use this format:

**value1=label1,value2,value3=label2**

Here is an example:

**0=Off,1,2=On**

This property is in the **Trace***N* property group.

#### trace*N*YAxisValueMax

Controls the range of y-axis if the traceNYAxisAutoScaleMode is set to **Off**. Select **On** for the traceNYAxisAutoScaleMode to calculate the y-axis range according to data values being plotted. To calculate a y-axis range that always includes traceNYAxisValueMin and `traceNYAxisValueMax`, select **On - Include Min/Max**.

This property is used only if yAxisMultiRangeMode is set to **Multiple Axis** or **Strip Chart**.

This property is in the **Trace***N* property group.

#### trace*N*YAxisValueMin

Controls the range of y-axis if the traceNYAxisAutoScaleMode is set to **Off**. Select **On** for the traceNYAxisAutoScaleMode to calculate the y-axis range according to data values being plotted. To calculate a y-axis range that always includes `traceNYAxisValueMin` and "traceNYAxisValueMax" on page 526, select **On - Include Min/Max**.

This property is used only if yAxisMultiRangeMode is set to **Multiple Axis** or **Strip Chart**.

This property is in the **Trace***N* property group.

## Trend graph: Trace Groups group

The properties in this group allow you to form trace groups. A trace group is a collection of two or more traces, and is useful for the following:

- Identifying multiple traces that should share one vertical axis, in strip chart or multi-axis modes
- Identifying three traces to be combined as a banded trace

By default, the category contains a single property, `traceGroupCount`, with a default value of zero. Legal values are 0 through 5. If nonzero, the `traceGroupNTraceNumbers` and `traceGroupNBandedFlag` properties appear in the Trace Group category, for each `groupN`, where *N* is between 1 and `traceGroupCount`, inclusive.

Follow these steps to construct an example:

1. Set `traceCount` = 3.
2. Set `traceGroupCount` = 1 (this makes the next 2 properties appear).
3. Set `traceGroup1TraceNumbers` = 1, 2, 3.
4. Set `traceGroup1BandedFlag` = true (checked).
5. Attach trace 1, 2, and 3 to data.

Note also the following:

- For each trace that is in a group, the y axis for the group is visible unless `traceNYAxisVisFlag` or `traceVisFlag` is false (unchecked) for all traces in the group. Similarly, the group's y axis grid is visible unless the `traceNYAxisGridVisFlag` property is false for all traces in the group.
- If `traceN` is in a group, these properties are hidden: `traceNYAxisAutoScaleMode`, `traceNYAxisValueMax`, `traceNYAxisValueMin`. The graph's `yAXisAutoScaleMode`, `yValueMax`, and `yValueMin` properties are used to scale each group's y axis range.
- If `yAxisMultiRangeMode` is Multiple Axis, the color of the axis for a group is determined by `traceNLineColor` of the first visible trace in the group. In Multiple Axis mode it may not be visually obvious which traces belong to which groups, unless you assign a similar line color or style, or mark color or style, to all the traces in a group.

### Trace Groups group properties

The group includes the following properties:


- "traceGroupNBandedFlag" on page 528
- "traceGroupNTraceNumbers" on page 528
- "traceGroupCount" on page 528

#### traceGroup*N*BandedFlag

If this property is checked, the group is expected to have three traces. The plot area beneath the first trace in the group (the low band trace) is filled, the second trace in the group (the value trace) is not filled, and the area above the third trace (the high band trace) is filled.

This property is in the **TraceGroups** property group.

#### traceGroup*N*TraceNumbers

Specifies a comma-separated list of the traces that belong to the group. If the trend graph is in strip chart mode or multi-axis mode, all the traces in the group share the same axis or strip.

This property is in the **TraceGroups** property group.

#### traceGroupCount

Sets the number of trace groups. The default is 0. Allowable values are 0 through 5. If nonzero, the properties `traceGroupNTraceNumbers` and `traceGroupNBandedFlag` appear in the Trace Group category, for each `groupN`, where *N* is between 1 and the value of `traceGroupCount`, inclusive.

This property is in the **TraceGroups** property group.

## Trend graph: X-Axis group

Properties in this group control the range and labeling of the x-axis.

#### X-Axis group properties

The group includes the following properties:


- "timeFormat" on page 529
- "timeRange" on page 529
- "timeRangeBegin" on page 529
- "timeRangeEnd" on page 529
- "timeRangeOfHistory" on page 530
- "timeShift" on page 530
- "xAxisFlag" on page 530
- "xAxisGridVisFlag" on page 530
- "xAxisLabelTextHeight" on page 530
- "xAxisMajorDivisions" on page 530
- "xAxisMinorDivisions" on page 530

#### timeFormat

Sets the format for the time displayed in the x-axis using syntax from the Java `SimpleDateFormat` class.

For example, **MMMM dd, yyyy hh:mm:ss a** results in dates of the form **August 30, 2003 05:32:12 PM**. If no format is given, the date and time are not displayed on the x-axis.

Include a new line character (`'\n'`) to display multiple-line text in the time axis labels. For example, **MM\dd'\n'hh:mm:ss** results in the following form:

```
08\30
05:32:12
```

If left blank, the axis is labeled with a default format based on the range.

This property is in the **X-Axis** property group.

#### timeRange

Sets the total amount of time, in seconds, plotted on the chart.

If `timeRange` is set to **-1**, the time range is determined by the first and last timestamp found in the traceNValue and traceNValueTable. If both tables are empty, the chart uses the first and last timestamp of the first overlay trace that has a non-empty `traceNValueTable` or `traceNValue`.

> **Note:** `timeRange` is ignored if both timeRangeBegin and timeRangeEnd are set.

The default is -1.0.

This property is in the **X-Axis** property group.

#### timeRangeBegin

Sets the start time value of the data to be plotted on the chart. Following are the supported formats:

- **mm/dd/yyyy hh:mm:ss** (for example, **01/16/2004 12:30:03**)
- **yyyy-mm-dd hh:mm:ss** (for example, **2004-01-16 12:30:03**)
- The number of milliseconds since midnight, January 1, 1970 UTC

> **Note:** If only the time is specified, the current date is used.

This property is in the **X-Axis** property group.

#### timeRangeEnd

Sets the end time value of the data to be plotted on the chart. Following are the supported formats are:

- **mm/dd/yyyy hh:mm:ss** (for example, **01/16/2004 12:30:03**)

- **yyyy-mm-dd hh:mm:ss** (for example, **2004-01-16 12:30:03**)
- The number of milliseconds since midnight, January 1, 1970 UTC

**Note:** If only the time is specified, the current date is used.

This property is in the **X-Axis** property group.

#### timeRangeOfHistory

Do not use this property

This property is in the **X-Axis** property group.

#### timeShift

Used to round the start and end times for the Y Axis. For example, if you specify -15 for the `timeShift` property, the start and end times for the Y Axis are rounded to the nearest 15 seconds.

The default value is -1.0.

This property is in the **X-Axis** property group.

#### xAxisFlag

Select to display the x-axis.

This property is in the **X-Axis** property group.

#### xAxisGridVisFlag

Set to display a vertical line for each major x-axis division.

This property is in the **X-Axis** property group.

#### xAxisLabelTextHeight

Specifies the height in pixels of the x-axis labels.

This property is in the **X-Axis** property group.

#### xAxisMajorDivisions

Specify the number of major divisions (long ticks) on the x-axis.

This property is in the **X-Axis** property group.

#### xAxisMinorDivisions

Specify the number of minor divisions (short ticks) on the x-axis.

This property is in the **X-Axis** property group.

## Trend graph: Y-Axis group

Properties in this group control the visibility and scaling of the y-axis or y-axes, as well as y-axis labeling and y-axis divisions. They also control the visibility of y-axis grid lines (but see also "Trend graph: TraceN group" on page 520).

### Y-Axis group properties

The group includes the following properties:


- "yAxisAutoScaleMode" on page 531
- "yAxisAutoScaleVisTracesOnlyFlag" on page 531
- "yAxisFlag" on page 532
- "yAxisGridVisFlag" on page 532
- "yAxisLabelTextHeight" on page 532
- "yAxisMajorDivisions" on page 532
- "yAxisMinLabelWidth" on page 532
- "yAxisMinorDivisions" on page 532
- "yAxisMultiRangeMode" on page 532
- "yAxisPosition" on page 533
- "yAxisValueLabels" on page 533


### yAxisAutoScaleMode

Controls how the y-axis range is calculated. Select one of the following from the drop down menu:

- **Off**: The yValueMin and yValueMax properties determine the range of y-axis.
- **On**: The dashboard calculates the y-axis range according to data values being plotted.
- **On - Include Min/Max**: The dashboard calculates the smallest range (with rounding) that includes yValueMin and yValueMax as well as all plotted points.

This property is in the **Y-Axis** property group.

### yAxisAutoScaleVisTracesOnlyFlag

Specifies that only visible traces should be used in scaling the y-axis when yAxisAutoScaleMode is not **OFF**. See traceNVisFlag.

This property is in the **Y-Axis** property group.

#### yAxisFlag

Controls the visibility of the labels and ticks for `trace01`, if yAxisMultiRangeMode is set to **Off** or **Classic**.

This property is in the **Y-Axis** property group.

#### yAxisGridVisFlag

Set to display a horizontal line for each major y-axis division for `trace01`, if yAxisMultiRangeMode is set to **Off** or **Classic**.

This property is in the **Y-Axis** property group.

#### yAxisLabelTextHeight

Specifies the height of the y-axis labels in pixels, if yAxisMultiRangeMode is set to **Off** or **Classic**

This property is in the **Y-Axis** property group.

#### yAxisMajorDivisions

Specifies the number of major divisions (wide ticks) on the y-axis, if yAxisMultiRangeMode is set to **Off** or **Classic**.

This property is in the **Y-Axis** property group.

#### yAxisMinLabelWidth

Specifies the minimum width of the y-axis labels in pixels, if yAxisMultiRangeMode is set to **Off** or **Classic**.

This property is in the **Y-Axis** property group.

#### yAxisMinorDivisions

Specifies the number of minor divisions (narrow ticks) on the y-axis, if yAxisMultiRangeMode is set to **Off** or **Classic**.

This property is in the **Y-Axis** property group.

#### yAxisMultiRangeMode

To specify the appearance of the y-axis when this chart has multiple traces, select one of the following from the drop down menu:

- **Off**: Only a single set of labels and ticks appears on the y-axis. The label range is based on the following properties: yAxisAutoScaleMode, yValueMin, and yValueMax.
- **Classic**: Ticks and labels for `trace01` appear on the left, outside of the plot area. Labels for the remaining traces appear on the left, inside of the plot area. With this setting for `yAxisMultiRangMode`, the "yAxisPosition" on page 533 property

is ignored. The label range for each trace is based on the following properties: `yAxisAutoScaleMode`, yValueMin, and yValueMax.

- **Multiple Axis**: A set of labels and ticks appears for each trace. The label range is determined independently for each trace, based on the following properties: traceNYAxisAutoScaleMode, traceNYAxisValueMin, traceNYAxisValueMax.
- **Strip Chart**: The y-axis is divided into sections, one for each trace. The traces are not overlaid, but rather appear one on top of the other. A set of labels and ticks appears for each trace. The label range is determined independently for each trace, based on the following properties: traceNYAxisAutoScaleMode, traceNYAxisValueMin, traceNYAxisValueMax.

This property is in the **Y-Axis** property group.

#### yAxisPosition

Specify the position of the y-axis ticks and labels. This property is ignored if yAxisMultiRangeMode is **Multiple Axis** or **Strip Chart**.

This property is in the **Y-Axis** property group.

#### yAxisValueLabels

Set to display a text label or tick mark on the y-axis in place of a numerical value, if yAxisMultiRangeMode is set to **Off** or **Classic**. Include a value with no label to display a tick mark without a label. Use this format:

```
value1=label1,value2,value3=label2
```

Here is an example:

```
0=Off,1,2=On
```

This property is in the **Y-Axis** property group.

# 14 Drill-Down Specification


- Using the Drill Down Properties dialog ..... 536
- Activating drill downs ..... 538
- About drilldown displays opened in Dashboard Builder ..... 538

The Dashboard Builder allows you to build customized display hierarchies by assigning *drill-down targets* to a dashboard's visualization objects. A given object's drill-down target is a dashboard that is displayed when the end user *activates* the drill down, typically by clicking on the given object.

## Using the Drill Down Properties dialog

In the Object Properties window, double-click on `drillDownTarget` in the **Property Name** field to bring up the Drill Down Properties dialog.

The Drill Down Properties dialog has the following fields and buttons:

- **Apply Drill Down To**: Choose one of the following from the drop down menu:
  - **New Window**: Open the targeted display file in a new display window. A new window is created each time this drill down is activated.
  - **Current Window**: Open the targeted display file in same window as the source object. With tabbed panels, open the targeted display in another tab if the display is already open; otherwise open in the selected tab.
  - **Named Window**: Open the targeted display file in a separate window defined by a specific name. The same window is reused each time this drill down is activated or if the end user activates another drill down with the same window name. If you choose this option, you must also enter a **Window Name**.
- **Window Name**: Enter a name for the window. The same window is reused for all drill down targets that reference this window name.

Entering **main** as a Window Name opens the targeted display in the top-level window. With multiple display panels, **main** opens the drill down in **panelcenter**.

> **Note:** This field is valid only if the drill down is applied to a **Named Window**.

- **Drill Down Display Name**: Select the name of the targeted display (`.rtv`) file. The drop down menu contains the names of files located in the current directory (typically, the `dashboards` directory of your project directory, under your Apama installation's work directory), as well as files located in first level of subdirectories. If a display is not listed, enter the name (including relative path) of the file. If the file path is a URL and it contains spaces, the spaces must be replaced with `%20`.

Select **Current Display** to target the display that is currently in the target window. This is most useful when **Current Display** is used in conjunction with **Current Window** or **Named Window**. Only substitutions specified in the Drill Down Properties dialog will be applied when the drill down is activated, and this allows you to use the source object to control data displayed by all objects in the window.

- **Drill Down Branch Function Name**: Enter the name of a function (in your current display) that returns the text string you want appended to the end of the **Drill Down Display Name**. This enables to you drill down to different displays based on the result of the function.

If the **Drill Down Display Name** is set to **Current Display** this option is not enabled.

- **Remove Existing Substitutions**: Select the **Remove Existing Substitutions** checkbox to remove existing drill down substitutions on the drill down window. This option is enabled only when you drill down to the **Current Window** or a **Named Window**.
- **Window Position**: Set the position of a new drill-down window. This option applies only when the drill down opens in a new window. Choose one of the following:
  - **Default**: Positioned by your operating system's window manager.
  - **Center of Screen**: Centered on the screen.
  - **Center of Parent**: Centered relative to the parent window.
  - **Relative to Screen**: Offset horizontally and vertically from the top left corner of the screen by the number of pixels specified in **Pixels from left** and **Pixels from top**.
  - **Relative to Parent**: Offset horizontally and vertically from the top left corner of the parent window by the number of pixels specified in **Pixels from left** and **Pixels from top**.
- **Window Title**: Specify text in the title bar. If this field is left blank, then the title bar will display a default title. This option applies only when the drill down opens in a new window.
- **Window Mode**: Specify modality and stacking order of drill down windows. This option applies only when the drill down opens in a new window.

There are three **Window Mode** options:

- **Normal**: Allow user interaction in all windows. Stacking order is determined by the **Drill Down Windows Always on Top** setting in the **General** tab of the **Application Options** dialog.
- **Modal**: Allow user interaction only in this drill down window while it is open. Stacking order is on top of all other dashboards.
- **Topmost**: Allow user interaction in any dashboard. Stacking order is on top of all other dashboards. Additionally, all windows targeted from a **Topmost** window will automatically assume the topmost position. *Note*: Some platforms do not support this functionality. If more than one window is set to be in the **Topmost** position, stacking order is platform dependent.

If this property is used on drill downs in Web-based, applet deployments, depending on your Security Manager, you may need to modify the Java security settings on each client to include the following permission:

```
permission java.awt.AWTPermission"setWindowAlwaysOnTop";
```

- **Drill Down Substitutions**: Direct the dashboard to assign values to specified dashboard substitution variables when the end user activates a drilldown on this object. *Note*: Some drill down substitutions are automatically added for displays targeted from table objects.

- **Add**: In the **String** field, enter a substitution variable. In the **Value** field, enter the value that you want assigned to the substitution variable. Click **Add** to insert the assignment into the listing.

**Note:** Substitution strings cannot contain the following:

- **| . tab space , ; = < > ' " & / \ { } [ ] ( )**
- **Remove**: Select a substitution from the list and click **Remove**.
- **OK**: Applies values and closes the dialog.
- **Clear**: Clears all fields. Removes drill down target (once **OK** is selected).
- **Cancel**: Closes the dialog with last values applied.
- **Help**: Opens the Help dialog.

## Activating drill downs

Once a drill down target has been assigned to an object, you can activate the drill down from a deployed dashboard or from the Dashboard Builder:

- Dashboard Builder: Double click on the object, or else right-click on it and select **Drill Down** from the popup menu.
- Web-based deployment: Single click on the object or else right click on it and select **Drill Down** from the popup menu.
- Local deployment: By default, single-click on the object or else right-click on it and select **Drill Down** from the popup menu. To override the default, select **Tools | Options** in the Builder (do this before you generate the deployment package), and uncheck **Single-Click for Drill Down and Commands** in the **General** tab. This allows the end user to use either a double click or a right click.

**Note:** If a command has been assigned to an object, then you must right-click and select **Drill Down** from the popup menu to activate the drill down.

## About drilldown displays opened in Dashboard Builder

With the Dashboard Builder, when you open a drill-down display in a **New Window** or a **Named Window**, the window is subject to the following restrictions:

- The window does not have menus or a toolbar.
- You cannot edit the properties of objects in the that are in these drill down displays, although you can view these properties.
- You cannot paste objects into such drill-down displays, although you can copy objects from such drill-down displays and paste them into the top-level display.

It is possible to double-click on objects within drill-down windows in order to activate further drill down displays.

# III Dashboard Function Reference


- Introduction to Dashboard Functions ..... 543
- Scalar Functions ..... 547
- Tabular Functions ..... 563
- Expression Syntax in Dashboard Functions ..... 617

This document provides reference information on Dashboard functions, which allow you to perform calculations, filtering, formatting, and other operations on correlator data. Instead of attaching a visualization object directly to a correlator data table or variable, you can specify data tables and variables as arguments to a Dashboard function, and then attach the visualization object to the function. Objects that are attached to functions update dynamically as the function arguments are updated, just as visualization objects that are attached directly to correlator data update dynamically as correlator data is updated.

The information in this book is organized as follows:

- Introduction to Dashboard functions, as well as the Dashboard Builder **Functions** panel and **Edit Function** dialog.
- Descriptions of functions that operate on and return numerical values or text strings.
- Descriptions of functions that operate on or return tabular data.

# 15 Introduction to Dashboard Functions

■ Working with functions ..... 544

Dashboard functions allow you to perform calculations, filtering, formatting, and other operations on correlator data. Instead of attaching a visualization object directly to a correlator data table or variable, you can specify data tables and variables as arguments to a dashboard function, and then attach the visualization object to the function.

## Working with functions

Follow these steps to add, edit, remove, or examine functions:

1. Open the dashboard file that contains (or will contain) the visualization object that you want to attach to a function.
2. In the Dashboard Builder, select **Functions** from the **Tools** menu. The **Functions** panel appears.

   The Functions panel has the following buttons:

   - **Add**: Adds a new function. Brings up the **Edit Function** dialog.
   - **Copy**: Copies the selected function. Brings up the Edit Function Name dialog.
   - **Edit**: Allows you to edit the selected function. Brings up the Edit Function dialog.
   - **Remove**: Removes the selected function.
   - **Result**: Brings up a dialog that displays the result of executing the selected function.
   - **References**: Brings up a dialog that lists all the objects that directly reference the selected function. When you choose an object in that list, it is selected according to its type. When you select a display object, Builder highlights the object in the drawing area. When you select a function, Builder brings up the dialog with the designated object selected.
3. To add or edit a function, click **Add** or **Edit** In the **Functions** panel. The **Edit Function** dialog appears.
4. Fill in the fields of the **Edit Function** dialog, and click **OK** (to apply the values and close the dialog) or **Apply** (to apply the values and leave the dialog open).

   The Edit Function dialog has the following fields:

   - **Function Name**: Specify a name that is unique among functions that have been added to the current dashboard file. The name must not contain spaces. The name **function** is not allowed.
   - **Function Type**: Select the function that you want to add and specify arguments for. The dropdown list includes built-in functions as well as user-defined functions (see "Using Dashboard Functions" on page 135).
   - Argument fields: Once you select a function for the **Function Type** field, the dialog is populated with the argument fields that are appropriate to the selected function. For each argument field, you can either enter a value or attach the argument to data. To attach the argument to data, right-click in the argument

field, select **Attach to Data**, and select a data source. An argument that has been attached to data is displayed in green. Double-click to edit the data attachment. Right-click and select **Attach to Data** to change the data source. Right-click and select **Detach from Data** to remove the attachment.

- **Description**: Include a description of any length. This description will be visible in the #**Attach to Function Data** dialog.

In the **Edit Function** dialog, when you are editing a function whose argument refers to another function, you can edit the referenced function without leaving the dialog. When you right-click an argument that contains a reference to a function, an additional item, **Edit Function,** appears in the popup menu. If you select it, the **Edit Function** dialog for that function replaces the current **Edit Function** dialog. If you have unsaved changes you are prompted to save or discard them, or cancel the operation. In addition a button, **Back**, appears in the **Edit Function** dialog that takes you back to the function you were previously editing.

Once a function has been added in this way (with arguments specified), you can use the **Attach to Function Data** dialog in order to attach it to properties of visualization objects in the current dashboard file. Objects that are attached to functions update dynamically as the function arguments are updated, just as visualization objects that are attached directly to correlator data update dynamically as correlator data is updated.

Note that the added functions can be edited only from within the dashboard file that was opened when the functions were added. In addition, a function is only available for use within the dashboard file that was opened when the function was added, or (for **public** functions) within a file that includes the dashboard file that was opened when the function was added.

The reference documentation on dashboard functions is divided into two sections:

- "Scalar Functions" on page 547: Describes built-in dashboard functions that operate on and return numerical values or text strings.
- "Tabular Functions" on page 563: Describes built-in dashboard functions that operate on or return tabular data.

See also "Using Dashboard Functions" on page 135.

# 16 Scalar Functions


- Add ..... 549
- Average ..... 549
- Boolean Expression ..... 549
- Concatenate ..... 550
- Correlator Time Format ..... 550
- Date Add ..... 551
- Date Ceiling ..... 551
- Date Compare ..... 552
- Date Difference ..... 552
- Date Floor ..... 553
- Date Format ..... 553
- Date Now ..... 554
- Delta ..... 554
- Divide ..... 554
- Duration ..... 555
- Evaluate Expression As Double ..... 555
- Evaluate Expression As String ..... 556
- Format Number ..... 556
- Get Substitution ..... 557
- Init Local Variable ..... 557
- isWindowsOS ..... 557
- Max ..... 557
- Min ..... 558
- Modulo ..... 558
- Multiply ..... 558
- Percent ..... 558

- Quick Set Sub ..... 559
- Replace All ..... 559
- Replace Value ..... 560
- Set Substitution ..... 560
- Set Substitutions By Lookup ..... 560
- Subtract ..... 561
- Validate Substitutions ..... 561

This section lists the Dashboard functions that operate on and return numerical values or text strings.

## Add

Returns the result of adding the two arguments.

#### Arguments

This function has the following arguments:

- **Argument1**: Numeric value to be added.
- **Argument2**: Numeric value to be added.

The function returns a numeric value.

## Average

Returns the average of the two arguments.

#### Arguments

This function has the following arguments:

- **Argument1**: Numeric value.
- **Argument2**: Numeric value.

The function returns a numeric value.

## Boolean Expression

Returns **1** (true) if the result of performing a specified comparison is true; returns **0** (false) otherwise.

#### Arguments

The function has the following arguments:

- **Value 1**: Text string that specifies the first value to compare.
- **Operator**: Text string that specifies the comparison operator. Supply one of the following:
  - **and**
  - **or**
  - **xor**
  - **=**

- **!=**
- **>**
- **<**
- **>=**
- **<=**

- **Value 2**: Text string that specifies the second value to compare.

This function returns a numerical value.

## Concatenate

Returns the result of combining the two arguments into a single text string.

#### Arguments

This function has the following arguments:

- **Value1**: Numeric value or text string.
- **Value2**: Numeric value or text string.

The function returns a text string.

## Correlator Time Format

Converts a correlator timestamp to either epoch time in milliseconds or the specified date/time format.

#### Arguments

- **Correlator Time**: Correlator timestamp that you want to convert. This argument is required. If it is not specified or if the specified value is invalid the string `"0"` is returned and any specified formatting is not applied.
- **Format**: Optional. Leave this field blank to convert the correlator timestamp to epoch time. The returned value is in milliseconds. Or, specify a date/time pattern to use to format the correlator timestamp. You can specify any pattern format supported by the Java class `SimpleDateFormat`. If you specify an invalid value the string `"0"` is returned.

A correlator timestamp is in seconds with a decimal point before the milliseconds, for example, `1043189336.2`.

This function returns a string.

## Date Add

Returns the result of adding the specified number (which may be negative) of date part intervals to the specified date, and returns a string representing the resulting date/time.

#### Arguments

This function has the following arguments:

- **Date**: Text string specifying the date to which is added the specified number of date parts. This must be either a formatted date/time string or a Java standard date/time argument in milliseconds from Jan 1, 1970.
- **Number**: Numeric value. The number of date parts to add to the specified date.
- **Date Part**: Text string specifying the date part, the specified number of which are to be added to the specified date. Specify **s**, **m**, **h**, **d**, **w**, **M**, **q** or **y** for seconds, minutes, hours, days, weeks, months, quarters, or years.
- **Date Format**: Text string specifying the format of the function result. Specify a pattern string suitable for use with the Java `SimpleDateFormat` class. For example, the format **MMMM dd, yyyy hh:mm:ss a** results in dates of the form exemplified by **August 30, 2003 05:32:12 PM**. If no **Date Format** is given, the string is returned in the form exemplified by **08/30/03 05:32 PM**. Use **q**, **qqq** or **qqqq** for short, medium or long versions of quarter notation. For example, **qqq-yyyy** results in a string of the form exemplified by **Qtr 1-2005**.

The function returns a text string.

## Date Ceiling

Returns the ceiling of **Date** with respect to **Date Part**. In other words, the function determines which **Date Part** interval contains the **Date**, and returns a string representing the start value of the next **Date Part** interval.

#### Arguments

This function has the following arguments:

- **Date**: Text string specifying the date whose ceiling is to be returned. This must be either a formatted date/time string or a Java standard date/time argument in milliseconds from Jan 1, 1970.
- **Date Part**: Text string specifying the date part with respect to which the specified date's ceiling is to be returned. Specify **s**, **m**, **h**, **d**, **w**, **M**, **q** or **y** for seconds, minutes, hours, days, weeks, months, quarters, or years.
- **Date Format**: Text string specifying the format of the function result. Specify a pattern string suitable for use with the Java `SimpleDateFormat` class. For example, the format **MMMM dd, yyyy hh:mm:ss a** results in dates of the form exemplified by **August**

**30, 2003 05:32:12 PM**. If no **Date Format** is given, the string is returned in the form exemplified by **08/30/03 05:32 PM**. Use **q**, **qqq** or **qqqq** for short, medium or long versions of quarter notation. For example, **qqq-yyyy** results in a string of the form exemplified by **Qtr 1-2005**.

The function returns a text string.

## Date Compare

Compares **Date 1** and **Date 2**, each rounded down to the nearest **Date Part.**

#### Return Value

If **Date 1** (rounded down to the nearest **Date Part)** is less than **Date 2** (rounded down to the nearest **Date Part)**, the function returns **-1**. If **Date 1** (rounded down to the nearest **Date Part)** is greater than **Date 2** (rounded down to the nearest **Date Part)**, the function returns **1**. If **Date 1** (rounded down to the nearest **Date Part)** equals **Date 2** (rounded down to the nearest **Date Part)**, the function returns **0**.

For example, comparing 08/30/03 05:32 PM to 08/30/03 04:47 PM with **Date Part** set to m (for minute resolution) returns 1, while setting Date Part to d (for day resolution) causes this function to return 0.

#### Arguments

This function has the following arguments:

- **Date 1**: Text string specifying one of the dates to be compared. It must be either a formatted date/time string or a Java standard date/time argument in milliseconds from Jan 1, 1970.
- **Date 2**: Text string specifying the other date to be compared. It must be either a formatted date/time string or a Java standard date/time argument in milliseconds from Jan 1, 1970.
- **Date Part:** Text string that controls the resolution of the comparison. Specify s, m, h, d, w, M, q or y for seconds, minutes, hours, days, weeks, months, quarters or years.

The function returns a number.

## Date Difference

Returns the integer number of **Date Part** intervals by which **Date 1** (rounded down to the nearest **Date Part**) is less than **Date 2** (rounded down to the nearest **Date Part**). For example, the difference between 05/12/05 05:32 PM and 05/15/05 04:47 PM with **Date Part** set to d (for day) returns 3.

#### Arguments

This function has the following arguments:

- **Date 1**: Text string specifying the earlier date. It must be either a formatted date/time string or a Java standard date/time argument in milliseconds from Jan 1, 1970.
- **Date 2**: Text string specifying the later date. It must be either a formatted date/time string or a Java standard date/time argument in milliseconds from Jan 1, 1970.
- **Date Part:** Text string that specifies the date part with respect to which the difference is to be calculated. Specify s, m, h, d, w, M, q or y for seconds, minutes, hours, days, weeks, months, quarters or years.

The function returns a number.

## Date Floor

Returns the floor of **Date** with respect to **Date Part**. In other words, the function determines which **Date Part** interval contains **Date**, and returns a string representing the starting date/time value of that interval.

#### Arguments

This function has the following arguments:

- **Date**: Text string specifying the date whose floor is to be returned. This must be either a formatted date/time string or a Java standard date/time argument in milliseconds from Jan 1, 1970.
- **Date Part**: Text string specifying the date part with respect to which the specified date's floor is to be returned. Specify **s**, **m**, **h**, **d**, **w**, **M**, **q** or **y** for seconds, minutes, hours, days, weeks, months, quarters, or years.
- **Date Format**: Text string specifying the format of the function result. Specify a pattern string suitable for use with the Java `SimpleDateFormat` class. For example, the format **MMMM dd, yyyy hh:mm:ss a** results in dates of the form exemplified by **August 30, 2003 05:32:12 PM**. If no **Date Format** is given, the string is returned in the form exemplified by **08/30/03 05:32 PM**. Use **q**, **qqq** or **qqqq** for short, medium or long versions of quarter notation. For example, **qqq-yyyy** results in a string of the form exemplified by **Qtr 1-2005**.

The function returns a text string.

## Date Format

Returns a string representing the specified date in the specified format.

#### Arguments

This function has the following arguments:

- **Date**: Text string specifying the date to be formatted. This must be either a formatted date/time string or a Java standard date/time argument in milliseconds from Jan 1, 1970.

- **Date Format**: Text string specifying the format of the function result. Specify a pattern string suitable for use with the Java `SimpleDateFormat` class. For example, the format **MMMM dd, yyyy hh:mm:ss a** results in dates of the form exemplified by **August 30, 2003 05:32:12 PM**. If no **Date Format** is given, the string is returned in the form exemplified by **08/30/03 05:32 PM**.

The function returns a text string.

## Date Now

Returns a string representing the current date and time in the specified format.

#### Arguments

This function has the following argument:

- **Date Format**: Text string specifying the format of the function result. Specify a pattern string suitable for use with the Java `SimpleDateFormat` class. For example, the format **MMMM dd, yyyy hh:mm:ss a** results in dates of the form exemplified by **August 30, 2003 05:32:12 PM**. If no **Date Format** is given, the string is returned in the form exemplified by **08/30/03 05:32 PM**.

The function returns a text string.

## Delta

Returns the rate of change of **Value** over the specified time interval.

#### Arguments

This function has the following arguments:

- **Value**: The numeric value whose rate of change is to be returned.
- **Interval**: Numeric value specifying the time interval, in seconds, for which the rate of change is to be calculated. If no value is given, the absolute delta is returned.

The function returns a number.

## Divide

Returns the result of dividing the first argument by the second.

#### Arguments

This function has the following arguments:

- **Argument1**: Numeric value specifying the dividend.
- **Argument2**: Numeric value specifying the divisor.

The function returns a number.

## Duration

Returns a string representing the specified duration in the specified format.

#### Arguments

This function has the following arguments:

- **Duration**: Numeric value specifying the duration, in milliseconds, to be formatted.
- **Duration Format**: Text string specifying the format of the function result. This string may contain 0 or more of the characters **d**, **s**, and **.** (for example **ds**) indicating that days, seconds, or milliseconds are to be included, in addition to hours and minutes, in the returned string. If no **Duration Format** is specified, the string is returned in the form exemplified by **15:32** (hours:minutes).

The function returns a text string.

## Evaluate Expression As Double

Returns the result of evaluating a specified expression that contains variables, each of which has an associated function argument. The result is returned as a double. Boolean true or false values are returned as **1.0** and **0.0** respectively.

#### Arguments

The function has the following arguments:

- **Expression**: Text string that specifies the expression to evaluate. Prefix variable names with **%**. Use standard arithmetic and logical operators. You can also use a variety of mathematical and string functions, as well as numeric and string constants. Enclose string constants in double quotes.
- Expression variable arguments: When the **Expression** field of the **Edit Function** dialog is activated (by pressing Enter or navigating to another field), the dialog displays a text field for each variable. For each field, enter a numeric value or text string. Values whose form is numeric are substituted into the expression as numbers; otherwise they are substituted into the expression as strings.

If a value whose form is numeric needs to be treated as a string, for example to serve as an argument to a string function, surround the variable in **Expression** with double quotes. Variables enclosed in double quotes are always used as strings. An example of such an expression is **length("%var1") + %var2**.

This function returns a numerical value.

## Evaluate Expression As String

Returns the result of evaluating a specified expression that contains variables, each of which has an associated function argument. The result is returned as a text string. Boolean true or false values are returned as **1.0** and **0.0** respectively.

#### Arguments

The function has the following arguments:

- **Expression**: Text string that specifies the expression to evaluate. Prefix variable names with **%**. Use standard arithmetic and logical operators. You can also use a variety of mathematical and string functions, as well as numeric and string constants. Enclose string constants in double quotes.
- Expression variable arguments: When the **Expression** field of the **Edit Function** dialog is activated (by pressing Enter or navigating to another field), the dialog displays a text field for each variable. For each field, enter a numeric value or text string. Values whose form is numeric are substituted into the expression as numbers; otherwise they are substituted into the expression as strings.

If a value whose form is numeric needs to be treated as a string, for example to serve as an argument to a string function, surround the variable in **Expression** with double quotes. Variables enclosed in double quotes are always used as strings. An example of such an expression is **length("%var1") + %var2**.

This function returns a text string.

## Format Number

Returns a string representing the specified number in the specified format.

For example, if **Number To Format** is **50**, and **Format** is **$**, the function returns **$50.00**.

#### Arguments

This function has the following arguments:

- **Number To Format**: Numeric value specifying the number to be formatted.
- **Format**: Text string specifying the format of the function result. The format can be specified based on the Java format specification, or with the following shorthand: **$** for US dollar money values, $$ for US dollar money values with additional formatting, or () for non-money values, formatted similar to money. Both positive and negative formats can be supplied, for example: #,###;(#,###).

The function returns a text string

## Get Substitution

Returns the current value of the given Substitution String.

#### Arguments

This function has the following argument:

- **Substitution String**: Text string specifying the substitution whose value is to be returned.

The function returns a text string.

## Init Local Variable

Initializes the local variable to the specified value. If the variable already has a non-empty value, this function does nothing.

#### Arguments

This function has the following arguments:

- **Variable Name**: Text string specifying the variable whose value is to be initialized.
- **Initial Value**: Numeric value or text string specifying the initial value to which the variable is to be set.

This function is useful for initializing a local variable to a value supplied by a data attachment.

## isWindowsOS

Returns 1 if the dashboard is not running in an applet and the operating system it is running on is Windows; returns 0 otherwise.

#### Arguments

This function has no arguments.

The function returns a number.

## Max

Returns larger of the two arguments.

#### Arguments

This function has the following arguments:

- **Argument1**: Numeric value.
- **Argument2**: Numeric value.

This function returns a number.

## Min

Returns smaller of the two arguments.

#### Arguments

This function has the following arguments:

- **Argument1**: Numeric value.
- **Argument2**: Numeric value.

This function returns a number.

## Modulo

Divides **Value** by **Divisor** and returns the remainder.

#### Arguments

This function has the following arguments:

- **Value**: Numeric value to be divided by **Divisor**.
- **Divisor**: Numeric value by which to divide **Value**.

This function returns a number.

## Multiply

Returns the result of multiplying the first argument by the second.

#### Arguments

This function has the following arguments:

- **Argument1**: Numeric value specifying one of the factors.
- **Argument2**: Numeric value specifying the other factor.

The function returns a number.

## Percent

Returns the percentage of **Value**, given the range defined by **Min Value** and **Max Value**.

#### Arguments

This function has the following arguments:

- **Value**: Numeric value specifying one of the factors.
- **Min Value**: Numeric value specifying one of the factors.
- **Max Value**: Numeric value specifying the other factor.

The function returns a number between 0 and 100.

## Quick Set Sub

Sets a substitution string to the specified value.

This function executes very quickly because, unlike the standard **Set Substitution** function, it does not search for and modify data attachments that use the substitution, it does not apply the change to child panels of the current panel, nor does it change the value of the local variable, if any, that is mapped to the substitution.

#### Arguments

This function has the following arguments:

- **Substitution String**: Text string specifying the substitution whose value is to be set.
- **Value**: Numeric value or text string specifying the value to which **Substitution String** is to be set.

This function is suitable for setting a substitution used only in a command or drilldown, but is not suitable for setting a substitution used in data attachments.

## Replace All

Replaces all occurrences of a given string which matches the pattern of the regular expression with another string. This function is just a wrapper of the java.lang.String.replaceAll() function. For detailed syntax, refer to the standard java.lang.String documentation.

If the **Substitution** argument is used, the replaced string is assigned to the substitution before it is returned.

#### Arguments

- **String**: String to be replaced.
- **Regular Expression**: Regular expression which defines the pattern of the string to be replaced.
- **Replacement**: String to be used as the replacement for the found pattern.

- **Substitution**: Optional. Name of the substitution (for example, `$symbol`) which is used for the assignment of the replaced string.

This function returns a string.

## Replace Value

Returns the replacement string that **Replacement Values** associates with **Value**.

For example, if **Value** is **Windows NT** and **Replacement Values** is '**Windows NT':winnt Windows2000:win2k**, the text string returned is **winnt**.

#### Arguments

This function has the following arguments:

- **Value**: Text string whose associated replacement string is to be returned.
- **Replacement Values**: Text string specifying value/replacement-string pairs. This is a space-separated list of pairs of the form *value*:*replacement-string*. Use a colon to separate the value from its associated replacement string. Use a space to separate one pair from another in the list. If a value or replacement string contains a space or colon, enclose that value or replacement string in single quotes.
- **Return Value if No Match**: Numerical value that controls what is returned if none of the pairs in **Replacement Values** has a value that matches **Value**. If **Return Value if No Match** is set to **1**, **Value** is returned when no match is found. If **Return Value if No Match** is set to **0**, the empty string is returned when no match is found.

The function returns a text string.

## Set Substitution

Sets the given substitution string to the given value, and returns the value.

#### Arguments

This function has the following arguments:

- **Substitution String**: Text string specifying the substitution whose value is to be set.
- **Value**: Numeric value or text string specifying the value to which **Substitution String** is to be set.

The function returns a text string.

## Set Substitutions By Lookup

Sets multiple substitutions based on the values in a specified lookup table.

#### Arguments

The function has the following arguments:

- **Key**: Text string or numeric value. This value identifies a row of **Lookup Table**, provided it matches a value in **Lookup Table**'s first column.
- **Lookup Table**: Table whose first column contains key values and whose remaining columns contain substitution values. These remaining columns have as names the names of substitution variables (and, in particular, they start with **$**).

**Key** is compared against the values in the first column of **Lookup Table** in order to determine which row of the lookup table to use to set substitution values. For each additional column in **Lookup Table** (where the column name starts with **$**), a substitution is set. The substitution name is the name of the column and the substitution value is the value from that column in the row whose first column matches **Key**.

This function returns a table.

## Subtract

Returns the result of subtracting the second argument from the first.

#### Arguments

This function has the following arguments:

- **Argument1**: Numeric value to be subtracted from.
- **Argument2**: Numeric value to subtract.

The function returns a number.

## Validate Substitutions

Validates a substitution string against the given table of valid values. Returns a substitution string with only valid values. The returned string is identical to the specified substitution string, except that any values from the specified string that are not found in the first column of the given table are replaced with the first value in the first column of the given table.

#### Arguments

The function has the following argument:

- **Substitution String**: Text string consisting a semicolon-separated list of substitutions (variable-name/value pairs) whose values are to be validated
- **Valid Value Table**: Table whose first column contains all values that are to be considered valid

- **Clear If Invalid**: If set to 1, the function returns an empty string if the substitution is not found in the table.
- **Allow Multiple Values**: If set to 1 this will allow the substitution to be a semicolon-separated string of values; each value will be tested for validity and the final result will be assembled from all the valid values (if any).

This function returns a text string.

# 17 Tabular Functions


- Add All Rows Or Columns ..... 565
- Add Columns ..... 566
- Average All Rows Or Columns ..... 567
- Average Columns ..... 568
- Baseline Over Time ..... 569
- Buffer Table Rows ..... 571
- Combine ..... 572
- Concatenate Columns ..... 573
- Convert Columns ..... 574
- Copy ..... 575
- Count ..... 576
- Count By Bands ..... 576
- Count Unique Values ..... 577
- Count Unique Values By Time ..... 577
- Create Selector List ..... 578
- Delta And Rate Rows ..... 579
- Delta Rows ..... 579
- Distinct Values ..... 580
- Divide Columns ..... 581
- Ensure Columns ..... 582
- Ensure Timestamp Column ..... 582
- Evaluate Expression By Row ..... 583
- Filter And Extract Matches ..... 585
- Filter By Pattern ..... 586
- Filter By Row ..... 586
- Filter By Time Range ..... 587

- First Table Rows ..... 587
- Format Table Columns ..... 588
- Get Data Server Connection Status ..... 588
- Group By Time ..... 589
- Group By Time and Unique Values ..... 590
- Group by Unique Values ..... 591
- Join ..... 596
- Join Outer ..... 598
- Last Table Rows ..... 599
- Mark Time Gaps ..... 599
- Max All Rows or Columns ..... 600
- Max Columns ..... 601
- Min All Rows or Columns ..... 602
- Min Columns ..... 603
- Modulo Columns ..... 604
- Multiply Columns ..... 605
- Percent Columns ..... 606
- Pivot On Unique Values ..... 607
- Reference ..... 609
- Rename Columns ..... 609
- Select Column ..... 610
- Set Column Type ..... 610
- Sort Table ..... 611
- Split String ..... 611
- String to Table ..... 611
- Subtotal By Time ..... 612
- Subtotal By Unique Values ..... 613
- Subtract Columns ..... 614
- Table Contains Values ..... 615

This section lists the Dashboard functions that operate on or return tabular data.

## Add All Rows Or Columns

Calculates the sum across cells for each row or column of the specified Table.

#### Usage notes

If **Return Column** is **1**, the function returns a table with one column. The *n*th cell of the returned column contains the sum of the numerical cells in the *n*th row of the table specified by the argument **Table**. (If there are no numerical cells in the row, the returned cell contains **0**.)

If **Return Column** is **0**, the function returns a table with one row. The *n*th cell of the returned row contains the sum of the numerical cells in the *n*th column of the table specified by the argument **Table**. (If there are no numerical cells in the column, the returned cell contains **N/A**, by default—but see the argument **Result Label Column**, below.) The *n*th column of the returned one-row table is labeled with the label of the *n*th column of the table specified by the argument **Table**.

#### Arguments

This function has the following arguments:

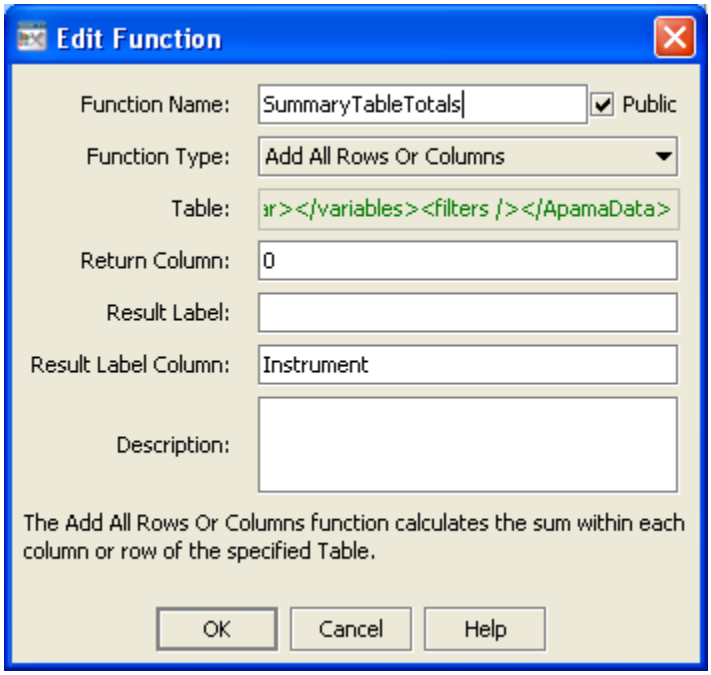
- **Table**: Table for which row or column sums are to be calculated.
- **Return Column**: Numeric value that controls whether to return a column or a row of result values. To get a column of result values, one value for each row, set **Return Column** to **1**. To get a row of result values, one value for each column, set **Return Column** to **0**.
- **Result Label**: Text string that specifies a label for the result row or column. If not specified, the label text is **Total**. If **Return Column** is **0**, the label appears only if **Result Label Column** is set to a column of **Table** that has no numeric values. If **Return Column** is **1**, the label text always appears and **Result Label Column** is ignored.
- **Result Label Column**: Text string that specifies the column in which **Result Label** appears, if **Return Column** is **0**. The specified column must have no numeric values in order for the label to appear. If **Return Column** is **1**, this argument is ignored.

This function returns a table.

#### Example

The second table below is attached to the function defined by the following dialog. The first table's data table is attached to the argument **Table**.

**Add All Rows Or Columns example**




Table

| Instrument | Price | Velocity | Shares | Position |
|---|---|---|---|---|
| MSFT | 27.21 | 0 | -800 | -21,768 |
| PRGS | 59.92 | 0 | -1000 | -59,920 |
| ORCL | 9.74 | 0 | 3000 | 29,220 |

Table

| Instrument | Price | Velocity | Shares | Position |
|---|---|---|---|---|
| Total | 96.8699... | 0 | 1200 | -52,468 |

## Add Columns

Returns a table that includes a column reflecting the sum of two specified columns of a specified table, the sum of a specified value and a specified column, or the sum of two specified values.

#### Usage notes

- Case 1: Sum of two specified columns of a specified table. This is the case if **First Column Name or Numeric Value** and **Second Column Name or Numeric Value** both specify a column of **Table**.
- Case 2: Sum of a specified value and a specified column. This is the case if one of **First Column Name or Numeric Value** and **Second Column Name or Numeric Value** specifies a column of **Table** and the other specifies a numeric value.
- Case 3: Sum of two specified values. This is the case if **First Column Name or Numeric Value** and **Second Column Name or Numeric Value** both specify a numeric value.

In case 1, the *n*th cell in the returned column corresponds to the *n*th cell of the table specified by the argument **Table**. Each cell of the returned column contains the result of adding the corresponding row's cell in **First Column Name or Numeric Value** to the cell in **Second Column Name or Numeric Value**.

In case 2, the *n*th cell in the returned column corresponds to the *n*th cell of the table specified by the argument **Table**. Each cell of the returned column contains the result of adding the corresponding row's cell in the specified column to the specified numeric value.

In case 3, each cell of the returned column contains the sum of the two specified numeric values.

In the returned table, the sum column is preceded by copies of all the columns in **Table**.

#### Arguments

The function has the following arguments:

- **Table**: Table that contains the columns to be summed.
- **First Column Name or Numeric Value**: Text string specifying the first column to be included in the sum, or numerical value to be included in the sum.
- **Second Column Name or Numeric Value**: Text string specifying the second column to be included in the sum, or numerical value to be included in the sum.
- **Result Column Name**: Text string that specifies the name of the column containing the sums. You must supply a value for this argument.

This function returns a table.

## Average All Rows Or Columns

Calculates the average across cells for each row or column of the specified Table.

#### Usage notes

If **Return Column** is **1**, the function returns a table with one column. The *n*th cell of the returned column contains the average of the numerical cells in the *n*th row of the table

specified by the argument **Table**. (If there are no numerical cells in the row, the returned cell contains **0**.)

If **Return Column** is **0**, the function returns a table with one row. The *n*th cell of the returned row contains the average of the numerical cells in the *n*th column of the table specified by the argument **Table**. (If there are no numerical cells in the column, the returned cell contains **N/A**, by default—but see the argument **Result Label Column**, below.) The *n*th column of the returned one-row table is labeled with the label of the *n*th column of the table specified by the argument **Table**.

#### Arguments

This function has the following arguments:

- **Table**: Table for which row or column averages are to be calculated.
- **Return Column**: Numeric value that controls whether to return a column or a row of result values. To get a column of result values, one value for each row, set **Return Column** to **1**. To get a row of result values, one value for each column, set **Return Column** to **0**.
- **Result Label**: Text string that specifies a label for the result row or column. If not specified, the label text is **Average**. If **Return Column** is **0**, the label appears only if **Result Label Column** is set to a column of **Table** that has no numeric values. If **Return Column** is **1**, the label text always appears and **Result Label Column** is ignored.
- **Result Label Column**: Text string that specifies the column in which **Result Label** appears, if **Return Column** is **0**. The specified column must have no numeric values in order for the label to appear. If **Return Column** is **1**, this argument is ignored.

This function returns a table.

## Average Columns

Returns a table that includes a column reflecting the average of two specified columns of a specified table, the average of a specified value and a specified column, or the average of two specified values.

#### Usage notes

- Case 1: Average of two specified columns of a specified table. This is the case if **First Column Name or Numeric Value** and **Second Column Name or Numeric Value** both specify a column of **Table**.
- Case 2: Average of a specified value and a specified column. This is the case if one of **First Column Name or Numeric Value** and **Second Column Name or Numeric Value** specifies a column of **Table** and the other specifies a numeric value.
- Case 3: Average of two specified values. This is the case if **First Column Name or Numeric Value** and **Second Column Name or Numeric Value** both specify a numeric value.

In case 1, the *n*th cell in the returned column corresponds to the *n*th cell of the table specified by the argument **Table**. Each cell of the returned column contains the result of

computing the average of the corresponding row's cell in **First Column Name or Numeric Value** and the cell in **Second Column Name or Numeric Value**.

In case 2, the *n*th cell in the returned column corresponds to the *n*th cell of the table specified by the argument **Table**. Each cell of the returned column contains the result of computing the average of the corresponding row's cell in the specified column and the specified numeric value.

In case 3, each cell of the returned column contains the average of the two specified numeric values.

In the returned table, the average column is preceded by copies of all the columns in **Table**.

#### Arguments

The function has the following arguments:

- **Table**: Table that contains the columns to be averaged.
- **First Column Name or Numeric Value**: Text string specifying the first column to be included in the average, or numerical value to be included in the average.
- **Second Column Name or Numeric Value**: Text string specifying the second column to be included in the average, or numerical value to be included in the average.
- **Result Column Name**: Text string that specifies the name of the column containing the averages. You must supply a value for this argument.

This function returns a table.

## Baseline Over Time

Calculates a baseline average of the values in the specified table over the specified number of specified date part intervals, and offsets the timestamp to a specified reference time.

#### Arguments

The function has the following arguments:

- **Table**: Data table for which the baseline is to be calculated. The specified table must contain a time column and a number column.
- **Date Part**: Text string specifying the date unit to use. Enter **s**, **m**, **h**, **d**, **w**, **M**, **q**, or **y**, for seconds, minutes, hours, days, weeks, months, quarters, or years. If left blank, the argument defaults to seconds.
- **Date Parts Per Interval**: Number of date parts in each interval over which the baseline is to be calculated.
- **Number Of Intervals**: Number of intervals over which the baseline is to be calculated. If this argument is set to **0**, the baseline is calculated over all the data in the table.

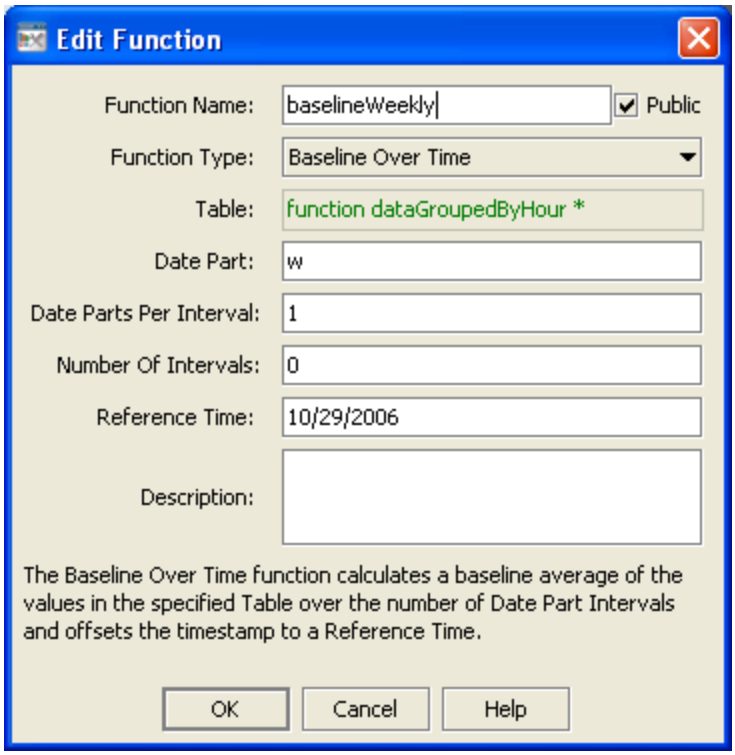
- **Reference Time**: After the baseline average has been calculated over the range of data specified, all values in the resulting time column are offset to start at the given reference time. This provides an easy way for the baseline to be plotted in a trend graph against a current set of values.

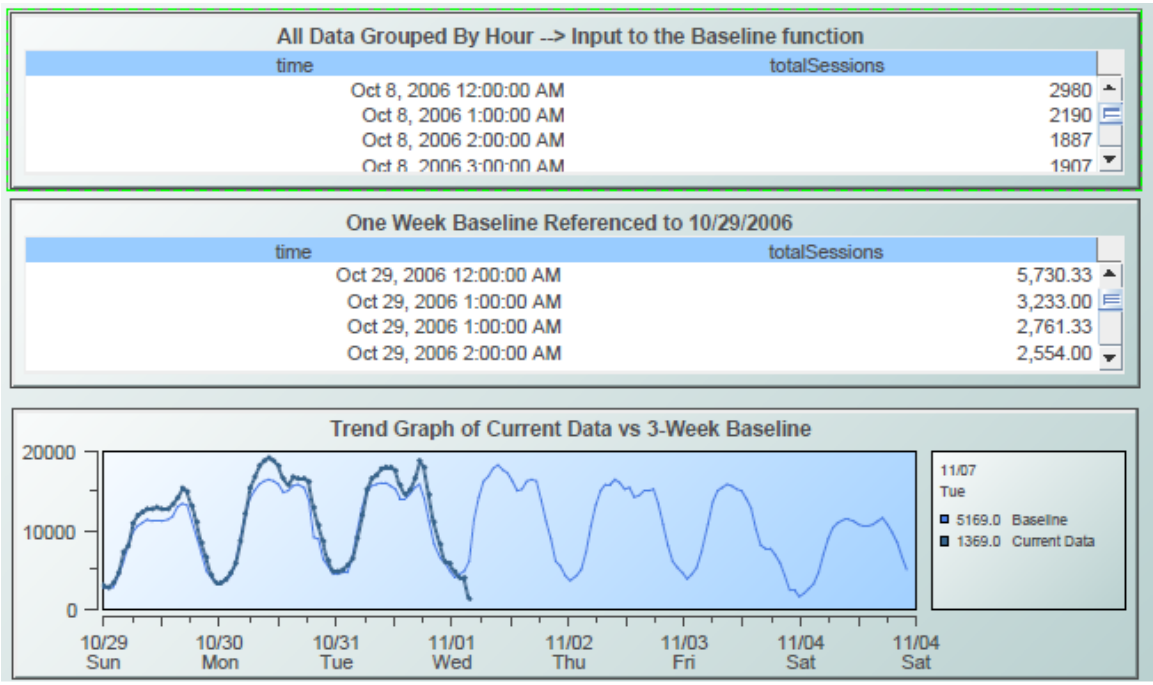
The function returns a table.

### Example

The trend graph below is attached to the function defined by the following dialog. In the trend graph, the thin, light blue line is the baseline, the average of the data over three one-week periods. The dark blue line is the current data. The first table's data table is attached to the argument **Table**. The second table shows the baseline data. Note that the current data for Sunday is lower than the rest of the current data. The graph shows that this is not anomalous, since the baseline data for Sunday is also lower than the rest of the baseline data.

### Baseline Over Time example






## Buffer Table Rows

Appends all rows of the input table to a buffer table that contains rows from previous updates. Returns the buffer table.

### Usage notes

This function is useful for buffering a table argument to another function in cases where the updates to the table may arrive rapidly (for example, from an event-driven data source), in order to ensure that the other function receives all rows.

### Arguments

The function has the following arguments:

- **Table**: Table whose rows are to be appended to the buffer table.
- **Number Of Rows**: Numeric value that specifies the number of rows in the returned buffer table. If necessary, older rows are removed to maintain this value.

**Note:** If the result of this function is used as the input to another function, all rows are removed from the buffer table after the other function is updated, regardless of value of **Number of Rows**.

This function returns a table.

## Combine

Returns the result of combining two specified tables into a single table.

#### Usage notes

When **Combine Rows** is **0**, the result contains the columns from **Table 1** followed by the columns from **Table 2**. Each result row consists of the *n*th row from **Table 1** followed by the *n*th row from **Table 2**. If **Table 1** and **Table 2** have a different number of rows, trailing result rows are padded with cells that are contain **0** or the empty string.

When **Combine Rows** is **1** and **Ignore Column Names** is **0**, the result contains the rows from **Table 1** followed by the rows from **Table 2**. The result table contains the column labels from **Table 1** followed by the column labels that appear only in **Table 2**. In the result table, **0** or the empty string appears in cells that are in rows from one table and in a column that appears only in the other table.

When **Combine Rows** is **1** and **Ignore Column Names** is **1**, the result contains the rows from **Table 1** followed by the rows from **Table 2**. The result table contains only columns that appear in both **Table 1** and **Table 2**.

#### Arguments

This function has the following arguments:

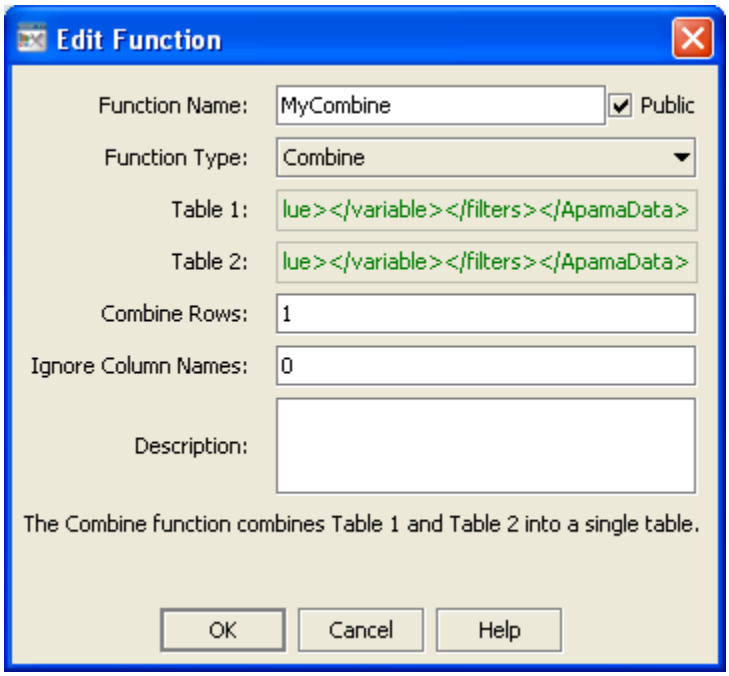
- **Table 1**: Table to be included in the combination operation.
- **Table 2**: Table to be included in the combination operation.
- **Combine Rows**: Numerical value that determines whether rows or columns are merged. When **Combine Rows** is **0**, the result contains the columns from **Table 1** followed by the columns from **Table 2**. When **Combine Rows** is **1**, the result contains the rows from **Table 1** followed by the rows from **Table 2**.
- **Ignore Column Names**: Numerical value that determines which columns are included in the result. When **Combine Rows** is **1** and **Ignore Column Names** is **1**, the result table contains only columns that appear in both **Table 1** and **Table 2**.When **Combine Rows** is **1** and **Ignore Column Names** is **0**, the result table contains the column labels from **Table 1** followed by the column labels that appear only in **Table 2**. This argument is ignored when **Combine Rows** is **0**.

This function returns a table.

#### Example

The third table below is attached to the function defined by the following dialog. The first table's data table is attached to the argument **Table 1**, and the second table's data table is attached to the argument **Table 1**.

**Combine example**




Table

| Instrument | Price | Shares | Position |
|---|---|---|---|
| MSFT | 26.62 | 11400 | 303,354 |
| PRGS | 59.51 | -2000 | -119,000 |

Table

| Instrument | Price | Velocity | Shares | Clip Size |
|---|---|---|---|---|
| ORCL | 9.55 | 0 | 4200 | 100 |

Table

| Instrument | Price | Shares | Position | Velocity | Clip Size |
|---|---|---|---|---|---|
| MSFT | 26.62 | 11400 | 303,354 | 0 | 0 |
| PRGS | 59.51 | -2000 | -119,000 | 0 | 0 |
| ORCL | 9.55 | 4200 | 0 | 0 | 100 |

## Concatenate Columns

Creates a string concatenation of the values in the given table columns separated by the given character(s), and returns the results in a new table column. The column names are

specified as a semicolon-separated string. The separator can be a single character such as . or / but it can also be a string such as **and**.

#### Arguments

This function has the following arguments:

- **Table**: Table whose column values are to be concatenated
- **Names of Columns to Concatenate**: Columns whose values are to be concatenated
- **Separating Character(s)**: Separator character, such as . or /, or separator string, such as **and**
- **Result Column Name**: Name of the result column

This function returns a table.

## Convert Columns

Returns a copy of the specified table that is modified by converting the specified columns to the specified types.

#### Usage notes

When converting from numeric types (other than **Long**) to the **Time** type, columns are first converted to **Long** and then to **Time**. If a **String** column entry cannot be parsed as a **Time**, the resulting entry is blank.

#### Arguments

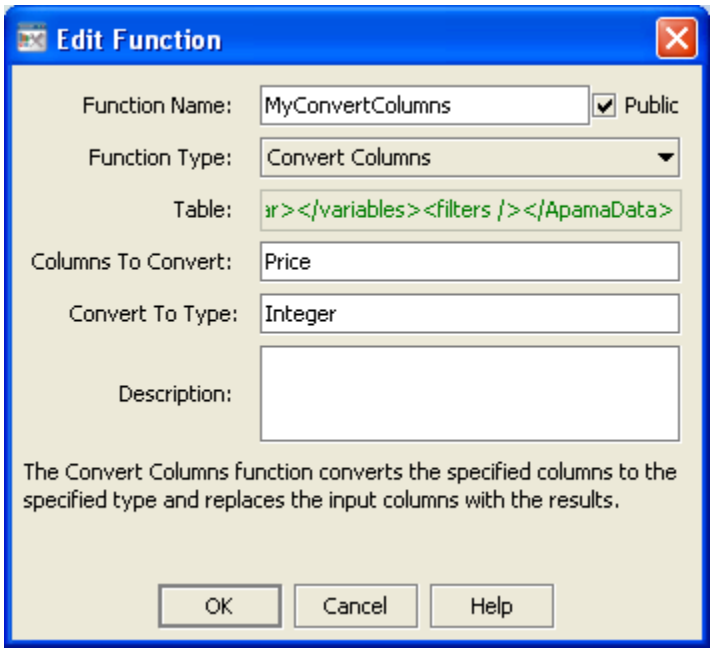
The function has the following arguments:

- **Table**: Table that contains the columns to convert.
- **Columns To Convert**: Text string that specifies the columns to convert. Supply a single column name or a semi-colon delimited list of column names.
- **Columns To Type**: Text string that specifies the target types of the conversion. Supply a single type name or a semi-colon delimited list of type names. Use the following type names: **Boolean**, **Integer**, **Long**, **Float**, **Double**, **String**, or **Time**. Type names may be abbreviated to the first letter.

This function returns a table.

The second table below is attached to the function defined by the following dialog. The first table's data table is attached to the argument **Table**.

### Convert columns example




Table

| Instrument | Price | Velocity | Shares | Position |
|---|---|---|---|---|
| MSFT | 27.02 | 0.0125 | 0 | 0 |
| PRGS | 60.23 | -0.0143 | -2200 | -132,506 |
| ORCL | 10.16 | 0 | -800 | -8,136 |

Table

| Instrument | Price | Velocity | Shares | Position |
|---|---|---|---|---|
| MSFT | 27 | 0.0125 | 0 | 0 |
| PRGS | 60 | -0.0143 | -2200 | -132,506 |
| ORCL | 10 | 0 | -800 | -8,136 |

## Copy

Copies the specified Table.

### Arguments

The function has the following argument:

- **Table**: Table to be copied.

This function returns a table.

## Count

Returns the number of rows in the specified table.

#### Arguments

The function has the following argument:

- **Table Column**: Table whose rows are to be counted.

This function returns a numeric value.

## Count By Bands

Divides a specified range of values into a specified number of bands and counts the number of rows in a specified data table column that contain a value that lies within each band.

#### Usage notes

If **Return Cumulative Percents** is set to **0**, this function returns a table containing one column that holds the midpoint values of each band (one row for each band), as well as one column containing the counts.

If **Return Cumulative Percents** is set to **1**, the returned columns will contain the cumulative percentage of the total count in each cell, rather than the individual counts.

#### Arguments

The function has the following arguments:

- **Table**: Data table column.
- **Number of Bands**: Numerical value that specifies the number of bands into which to divide the specified range.
- **Include Min/Max**: Numerical value (**0** for false and **1** for true) that determines whether the range of values is specified by **Min Value** and **Max Value**, or by the values in **Table**.
- **Min Value**: Numerical value that, together with **Max Value**, specifies the range of values that is divided into the bands, if **Include Min/Max** is **1**.
- **Max Value**: Numerical value that, together with **Min Value**, specifies the range of values that is divided into the bands, if **Include Min/Max** is **1**.
- **Return Cumulative Percent**: Numerical value (**0** or **1**) that determines whether counts or cumulative percentages are returned. If set to **1**, he function returns the cumulative percentage of the total count in each cell, rather than the individual counts.

This function returns a table.

## Count Unique Values

Returns a table that lists unique values and their counts from the specified table column.

#### Arguments

The function has the following arguments:

- **Table Column**: Data table column whose values are to be counted.
- **Value List**: Table column that specifies values for which a count is to be performed. If you do not supply this argument, counts are returned only for values present in **Table Column**. Use this argument to include rows in the returned table for values that are not always present in **Table Column**. If you specify this argument, the returned table includes a row for each specified value, even if the count for some values is 0.
- **Restrict to Value List**: Numerical value (0 or 1) that determines whether a count is performed *only* for values in **Value List**. If **Restrict to Value List** is set to 0, all unique values from the **Table Column** are included in the returned table. If **Restrict to Value List** is set to 1 and **Value List** is specified, only rows from the **Value List** are included.
- **Use Column Names**: Numerical value (0 or 1) that determines whether original column names are retained in the returned table. If **Use Column Names** is set to 1, then original column names are retained. If set to 0, columns are given generic names (for example, **Subtotal1** and **Total1**). Generic column names are useful when the data attachment for the **Table** argument uses a substitution that causes the column names to change when the substitution changes.

This function returns a table.

## Count Unique Values By Time

Returns a table that lists unique values and their counts from a specified table, sorted by a specified number of specified date part Intervals. The Table must contain a time column and a value column. The returned table contains an interval column, a column for each unique value, and counts for number of intervals specified or for all data in the Table if the Number of Intervals is **0**.

#### Arguments

The function has the following arguments:

- **Table**: Data table column whose values are to be counted.
- **Date Parts Per Interval**: Number of date parts in each interval by which the counts are to be sorted.
- **Number Of Intervals**: Number of intervals by which the counts are to be sorted.

- **Date Part**: Text string that specifies the date unit to use. Enter **s**, **m**, **h**, **d**, **w**, **M**, **q**, or **y**, for seconds, minutes, hours, days, weeks, months, quarters, or years.
- **Date Format**: Text string that specifies the format of the function result. Specify a pattern string suitable for use with the Java `SimpleDateFormat` class.
- **Value List**: Table column that specifies values for which a count is to be performed. If you do not supply this argument, counts are returned only for values present in **Table**. Use this argument to include rows in the returned table for values that are not always present in **Table**. If you specify this argument, the returned table includes a row for each specified value, even if the count for some values is **0**.
- **Restrict to Value List**: Numerical value (**0** or **1**) that determines whether a count is performed *only* for values in **Value List**. If **Restrict to Value List** is set to **0**, all unique values from the **Table Column** are included in the returned table. If **Restrict to Value List** is set to **1** and **Value List** is specified, only rows from the **Value List** are included.
- **Use Column Names**: Numerical value (**0** or **1**) that determines whether original column names are retained in the returned table. If **Use Column Names** is set to **1**, original column names are retained. If set to **0**, columns are given generic names (for example, **Subtotal1** and **Total1**). Generic column names are useful when the data attachment for the **Table** argument uses a substitution that causes the column names to change when the substitution changes.

This function returns a table.

## Create Selector List

Returns a two column table containing selector names and their corresponding values to be presented in a dropdown list. The first column contains selector names and the second column contains their values. The returned table consists of the first two columns of a specified table, optionally modified by sorting, and with the optional addition of a row that contains a specified selector name in the first column and the value * in the second column. If the input table has only one column its contents are used for both the selector names and values.

#### Arguments

This function has the following arguments:

- **Selector Table**: Table whose data is to be presented in a dropdown list.
- **All Selector Name:** Text string that controls whether an initial row is added to the returned table that contains. If you specify a value, a row is added whose first column contains the specified value and whose second column contains the value *****.
- **Sort Values**: Numeric value that controls whether the returned rows are sorted. Set this argument to **1** in order to sort the returned rows by selector name. The sort is numerical, if all the selector names are numbers; otherwise the sort is alphabetical.

- **Sort Descending**: Numeric value that controls whether returned rows are sorted in descending order, if the argument **Sort Values** is set to **1**.Set **Sort Descending** to **1** in order to sort selector names in descending order. Set the argument to **0** (or leave it blank)) in order to sort selector names in ascending order.

This function returns a table.

## Delta And Rate Rows

Returns a table that includes a rate-of-change column as well as a delta column for each specified data column. The function returns a table including, for the specified columns, new values for the difference between this update and the previous, along with the rate of change per second. The new values may be appended to the input table in columns named by prefixing **Delta** and **Rate** to the column name, or the delta columns may replace the corresponding input columns (the rate columns will still be appended to the table).

This function has the following arguments:

- **Table**: Table of interest
- **Delta Column Names**: Names of one or more columns for which deltas will be calculated. At least one name must be given.
- **Index Column Names**: Names of one or more columns that uniquely identify a row in the table. If left blank, the default is to calculate deltas for all rows as if they had the same value. The values contained in each index column are concatenated to form a unique index used to organize the resulting summary data..
- **Time Column Name**: Name of a timestamp column that will be used to calculate the rate of change. A name must be given. If the specified column is not found in the data it will be added, and its values will be taken from the current time on each update.
- **Replace Data With Deltas**: If set to 1, the delta values replace the original values in the same column in the returned table; otherwise they are in new columns appended to the table.
- **Display Negative values**: If set to 1, the delta values less than zero will be displayed with a negative sign and the value; otherwise they will be displayed as zero.

This function returns a table.

## Delta Rows

Computes the delta between incoming rows of tabular data. This function returns a table that includes, for the specified columns, new values for the difference between the current update and the previous update.

#### Arguments

The function has the following arguments:

- Table: Table for which
- **Delta Column Names**: Text string that specifies the name of one or more columns for which deltas are to be calculated. Separate column names by a semicolon. This field cannot be left blank.
- **Index Column Names**: Text string that specifies the name of one or more columns that uniquely identify a row in the table. Separate column names by a semicolon. If left blank, the function calculates deltas for all rows as if they had the same value. The values contained in each index column are concatenated to form a unique index that is used to organize the resulting summary data.
- **Replace Data With Deltas**: Numerical value (**0** or **1**) that determines whether the returned table includes columns with the original values from **Table**. If set to 1, delta values replace original values in columns with the original names. If set to 0, new columns are added. The new columns use the original columns names with **Delta** prefixed.
- **Display Negative Values**: If set to 1, delta values less than zero are displayed with a negative sign. If set to 0, delta values less than zero are displayed as zero.

This function returns a table.

## Distinct Values

Returns a table with a single column that lists all unique values from a specified column of a specified table.

#### Arguments

This function has the following arguments:

- **Table**: Table that contains the column whose unique values are to be returned
- **Column Name**: Name of the column whose unique values are to be returned
- **Sort Values**: Numeric value that controls whether the returned values are sorted. Set this to **1** in order to sort the values. Values are sorted in numerical order, if all values are numbers; otherwise they are sorted alphabetically.
- **Sort Descending**: Numeric value that controls whether the returned values are sorted in descending order, if **Sort Values** is set to **1**. Set **Sort Descending** to **1** in order to sort the values in descending order. Set the argument to **0** (or leave it blank) in order to sort the values in ascending order.
- **Use Column Name**: Numeric value that controls the label of the returned column. Set this argument to **1** in order to use the original column name (the value of the **Column Name** argument). Set the argument to **0** (or leave it blank) in order to use the name **Values**. Use a generic name when you want a display that is independent of the

value of **Table**, for example, because the value of **Table** uses a substitution that causes column names to change when the substitution changes.

This function returns a table.

## Divide Columns

Returns a table that includes a column reflecting the quotient of specified columns of a specified table, the quotient of a specified value and a specified column, or the quotient of two specified values.

#### Usage notes

- Case 1: Quotient of specified columns of a specified table. This is the case if **First Column Name or Numeric Value** and **Second Column Name or Numeric Value** both specify a column of **Table**.
- Case 2: Quotient of a specified value and a specified column. This is the case if one of **First Column Name or Numeric Value** and **Second Column Name or Numeric Value** specifies a column of **Table** and the other specifies a numeric value.
- Case 3: Quotient of two specified values. This is the case if **First Column Name or Numeric Value** and **Second Column Name or Numeric Value** both specify a numeric value.

In case 1, the *n*th cell in the returned column corresponds to the *n*th cell of the table specified by the argument **Table**. Each cell of the returned column contains the result of dividing the corresponding row's cell in **First Column Name or Numeric Value** by the cell in **Second Column Name or Numeric Value**.

In case 2, the *n*th cell in the returned column corresponds to the *n*th cell of the table specified by the argument **Table**. Each cell of the returned column contains the result of dividing the corresponding row's cell in the specified column by the specified numeric value.

In case 3, each cell of the returned column contains the quotient of the two specified numeric values.

In the returned table, the quotient column is preceded by copies of all the columns in **Table**.

#### Arguments

The function has the following arguments:

- **Table**: Table that contains the columns to be divided.
- **First Column Name or Numeric Value**: Text string specifying the column to whose values are to be used as dividends, or numerical value to be as dividends.
- **Second Column Name or Numeric Value**: Text string specifying the column to whose values are to be used as divisors, or numerical value to be as divisors.

- **Result Column Name**: Text string that specifies the name of the column containing the quotients. You must supply a value for this argument.

This function returns a table.

## Ensure Columns

Returns a copy of a specified table, modified where necessary to guarantee that given columns columns have specified types.

#### Arguments

The function has the following arguments:

- **Table**: The specified table.
- **Column Name(s)**: Text string that specifies the columns to be modified if necessary. Supply a single column name or a semicolon-separated list of column names.
- **Column Type(s)**: Text string that specifies the types. Supply a single name or a semicolon-separated list of names. The *n*th element of **Column Type(s)** is the type to be guaranteed for the column specified by the *n*th element of **Column Name(s)**.
- **Values**: Semicolon-separated list of values to substitute when a value must be modified. The *n*th element of **Values** is used for the column named by the *n*th element of **Column Name(s)**.

This function returns a table.

## Ensure Timestamp Column

Returns a copy of a specified table, supplemented if necessary to include a timestamp column with a specified name. The added column is filled with the current time.

#### Arguments

The function has the following arguments:

- **Table**: The specified table.
- **Column Name**: Name to be used for the timestamp column, if one is added.
- **Append Column**: If set to 1, the timestamp column is appended to the end of the table; otherwise it is inserted as the first column.

This function returns a table.

## Evaluate Expression By Row

Returns the result of evaluating a specified expression for each row of a specified table. The specified expression contains variables, each of which has an associated column of the specified table.

#### Usage notes

The returned table has all the columns of the specified table, followed by a column that contains the evaluation results. The *n*th row of the results column contains the result evaluating the expression with values from the *n*th row of the specified table substituted for expression variables.

Boolean true and false values are returned as **1.0** and **0.0** respectively.

#### Arguments

The function has the following arguments:

- **Expression**: Text string that specifies the expression to evaluate. Prefix variable names with **%**. Use standard arithmetic and logical operators. You can also use a variety of mathematical and string functions, as well as numeric and string constants. Enclose string constants in double quotes.
- **Table**: The table whose data is to be substituted into the **Expression** for evaluation.
- **Result Column Name**: Text string that specifies the name of the result column.
- **Result Column Type**: Text string that specifies the type of values in the result column. Specify either **double** or **string** (or the abbreviations **d** or **s**).
- Expression variable arguments: When the **Expression** field of the **Edit Function** dialog is activated (by pressing Enter or navigating to another field), the dialog displays a text field for each variable. For each field, enter a text string that names a column of **Table**. Column values are substituted for the corresponding variables in **Expression**. The types of the values are taken from the types of the columns. Numeric and boolean values are converted to **double**. **Date** columns are not supported.

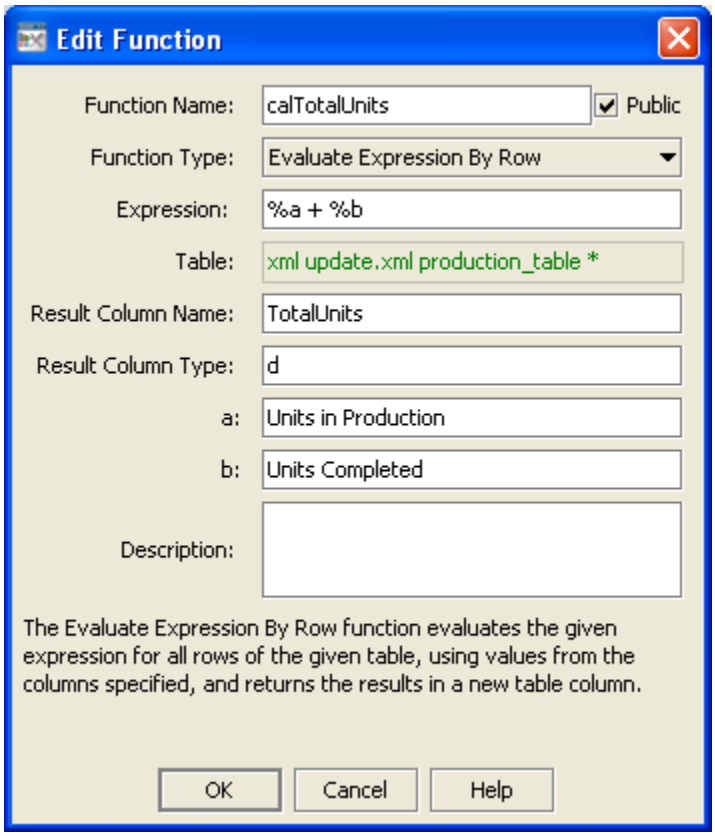
If a value whose form is numeric needs to be treated as a string, for example to serve as an argument to a string function, surround the variable in **Expression** with double quotes. Variables enclosed in double quotes are always used as strings. An example of such an expression is **length("%var1") + %var2**.

This function returns a text string.

#### Example

The second table below is attached to the function defined by the following dialog. The first table's data table is attached to the argument **Table**.

### Evaluate Expression By Row example




| Plant | Units in Prod... | Units Compl... | Status | On Schedul |
|---|---|---|---|---|
| San Francisco | 96 | 76 | waiting for s... | ☐ |
| San Jose | 82 | 43 | waiting for s... | ☑ |
| Dallas | 88 | 95 | waiting for s... | ☐ |
| Chicago | 85 | 100 | waiting for s... | ☑ |
| New York | 82 | 79 | online | ☐ |
| Detroit | 27 | 96 | online | ☑ |
| Baltimore | 28 | 25 | online | ☑ |
| New Orleans | 87 | 28 | online | ☑ |
| Seattle | 41 | 65 | online | ☑ |
| Denver | 61 | 13 | offline | ☑ |
| San Francisco | 98 | 95 | waiting for s... | ☐ |
| San Jose | 19 | 48 | waiting for s... | ☐ |
| Dallas | 91 | 13 | waiting for s... | ☑ |

| Units Completed | Units in Production | TotalUnits |
|---|---|---|
| 48 | 19 | 67 |
| 96 | 27 | 123 |
| 25 | 28 | 53 |
| 65 | 41 | 106 |
| 13 | 61 | 74 |
| 42 | 71 | 113 |
| 73 | 73 | 146 |

## Filter And Extract Matches

Returns a table containing all rows from a specified table in which the value of a specified column matches a specified pattern. For each matching row, each token from the specified column that matches a group in the pattern is extracted to a new column.

### Arguments

This function has the following arguments:

- **Table**: Table from which matching rows are to be extracted.
- **Filter Column Name**: Text string specifying the column of **Table** to be searched for matches.
- **Pattern**: Text string that is either a simple string that optionally uses * as a wildcard, or a regular expression as described in the following Web page:

  See http://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html
- **Pattern Is Reg Expr**: Numerical value (0 or 1) that determines whether the pattern is interpreted as a simple string (that optionally uses * as a wildcard) or as a regular expression. If this argument is 0 (the default), **Pattern** is interpreted as a simple string. Otherwise, **Pattern** is interpreted as a regular expression.
- **Number of New Columns**: Numerical value that specifies the number of new columns to be added to the result table to contain the matching groups extracted from the filter column.
- **New Column Names**: Text string that specifies the name of each new column. Separate column names with a semicolon.

This function returns a table.

### Example

Consider the following arguments:

- **Table**: Table that includes a **Customer Name** column.
- **Filter Column Name**: **CustomerName**.
- **Pattern**: *** ***
- **Pattern Is Reg Expr**: **0**
- **Number of New Columns**: 2
- **New Column Names**: **FirstName;LastName**

In this case, if a row of **Table** contains **Joe Smith** in the **CustomerName** column, the corresponding row in the result table contains **Joe** in the **FirstName** column and **Smith** in the **LastName** column.

## Filter By Pattern

Returns a table containing all rows from a specified table in which the value of a specified column matches a specified pattern.

#### Arguments

This function has the following arguments:

- **Table**: Table from which matching rows are to be extracted.
- **Filter Column Name**: Text string specifying the column of **Table** to be searched for matches.
- **Pattern**: Text string that is either a simple string that optionally uses * as a wildcard, or a regular expression as described in the following Web page:

  http://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html
- **Pattern Is Reg Expr**: Numerical value (0 or 1) that determines whether the pattern is interpreted as a simple string (that optionally uses * as a wildcard) or as a regular expression. If this argument is 0 (the default), **Pattern** is interpreted as a simple string. Otherwise, **Pattern** is interpreted as a regular expression.

This function returns a table.

## Filter By Row

Returns a table containing all rows from a specified table in which the values of specified columns matches the values in specified lists of values.

#### Arguments

This function has the following arguments:

- **Table**: Table from which matching rows are to be extracted.
- **Filter Column Name**: Text string that specifies a list of column names, the columns of **Table** to be searched for matches. Separate column names with
- **Filter Value**: Text string that specifies a *list of lists* of values to be matched. Separate values with a coma. Separate lists with a semicolon (for example, **val1,val2;val3,val4;val5,val6**). Enter ***** for **Filter Value** in order to display all rows in the table. To use ***** as a literal comparative value, enclose it in single quotes. To use **;** as a literal comparative value, enclose it in single quotes. If a filter value contains a space or a semicolon, enclose the entire value in single quotes.

A row from **Table** is included in the returned table if and only if the $n^{th}$ **Filter Column Name** matches a value in **FilterValue**'s $n^{th}$ list, for all $n$ from 1 to the number of specified column names.

This function returns a table.

## Filter By Time Range

Returns a copy of a specified table that contains only those rows in which the value of a specified column falls within a specified time range.

#### Arguments

The function has the following arguments:

- **Table**: Table whose rows are to be copied.
- **Date/Time Column Name**: Text string that specifies a column of **Table** that contains a timestamp. If this argument is not supplied, the first column of **Table** is assumed to contain a timestamp.
- **Time Range Start**: Text string that specifies the start of the desired time range. If this argument is not supplied, the time range is unbounded at the lower end.
- **Time Range End**: Text string that specifies the end of the desired time range. The time range itself does not include this value, but does include a value that is one second less than **Time Range End**. If this argument is not supplied, the time range is unbounded at the upper end.

This function returns a table.

## First Table Rows

Returns a table containing one of the following:

- First *n* rows of a specified table, for a specified number, *n*.
- First *n* rows for each unique combination of values in a specified set of columns in a specified table, for a specified number, *n*.

#### Arguments

This function has the following arguments:

- **Table**: Table some of whose rows are to be returned.
- **Index Column Names**: Text string that specifies the column or columns to be used to form indexes. Separate column names with a semicolon. If this argument is not supplied, the function returns the first **Number of Rows** of **Table**. If this argument is supplied, the function forms indexes by concatenating the values contained in each index column, and returns **Number of Rows** for each index.
- **Number of Rows**: Numerical value that specifies the number rows to be returned, or the number of rows with each index value to be returned.

The function returns a table.

## Format Table Columns

Returns a copy of a specified table with specified formats applied to specified columns.

#### Arguments

This function has the following arguments:

- **Table to Format**: Table whose columns are to be formatted.
- **Column Format(s)**: Text string that specifies the columns to format and the formats to use. The string consists of *column-name*:*column-format*. Separate pairs with a semicolon. Separate column name from column format with a colon. Enclose a column name in single quotes if it contains a space.

Specify the column format based on the Java format specification, or with the following shorthand: **$** for money values, **$$** for money values with additional formatting, or **()** for non-money values, formatted similar to money. For example, if **Column Format(s)** contains the pair **'Units Completed':$**, the **Units Completed** column in the returned table is formatted for money. Both positive and negative formats can be supplied, for example: **#,###;(#,###)**.

The function accepts date/time patterns for formatting columns of type Date. For example consider a column **Timestamp** with the value **Sep 06, 2008 7:27:36 AM**. If it is formatted with **'Timestamp':'MM/dd/yyyy'** the result is **09/06 /2008**. If it is formatted with **'Timestamp':'hh:mm:ss'** the result is **07:27:36**. For the full list of formatting codes, see the Java documentation for the class **SimpleDateFormat**.

The function returns a table.

## Get Data Server Connection Status

Returns a table that contains status information for the Data Server being used by the current dashboard.

#### Columns

The table has the following columns:

- **Name**: `__default` for the default Data Server, or the name of a named Data Server
- **Connected**: `True` if the server connection is operational; `False` otherwise
- **Status**: `OK` if connection is operational, `no connection` if there is no connection to the server, or `no service` if there is an HTTP connection to the `rtvdata` servlet but the servlet has no connection to its Data Server
- **ConnectionString**: URL for an HTTP connection to the `rtvdata` servlet or *hostname*:*port* for a direct socket connection to a Data Server
- **ReceiveCount**: Number of data transmissions (pushes) received from the server
- **ReceiveTime**: Time of the most recent data transmission from the server

- **Config**: String that identifies Data Server version

#### Arguments

The function has no arguments.

This function returns a table.

## Group By Time

Returns a table that contains a summary of all the data in a given table, aggregated over time. The summary data in the returned table is grouped into a specified number of specified time intervals over a specified time range.

#### Arguments

The function has the following arguments:

- **Table**: Table whose data is to be summarized.
- **Group Type**: Text string that specifies the type of aggregation to perform. Enter one or more of the following: **sum**, **count**, **average**, **min**, **max**. The default is **sum**. For multiple group types, use a semicolon-separated list and set **Use Column Names** to **0**.
- **Date/Time Column Name**: Text string that specifies a column of **Table** that contains a timestamp. If this argument is not supplied, the first column of **Table** is assumed to contain a timestamp.
- **Date Part**: Text string that specifies the date unit to use. Enter **s**, **m**, **h**, **d**, **w**, **M**, **q**, or **y**, for seconds, minutes, hours, days, weeks, months, quarters, or years. The default unit is seconds.
- **Date Parts Per Interval**: Numerical value that specifies he size of each interval in **Date Parts**.
- **Number Of Intervals**: Numerical value that specifies the number of intervals to include in the summary table. The returned table contains one row for each interval. If set to **0**, the number of intervals is determined from the range of data in **Table**, or by the specified time range.
- **Time Range Start**: Text string that specifies the start of the desired time range. If this argument is not supplied, the time range is unbounded at the lower end.
- **Time Range End**: Text string that specifies the end of the desired time range. The time range itself does not include this value, but does include a value that is one second less than **Time Range End**. If this argument is not supplied, the time range is unbounded at the upper end.
- **Restrict To Time Range**: Numerical value (**0** or **1**) that determines whether **Time Range Start** and **Time Range End** are ignored. If set to **1**, the resulting summary table includes only those time intervals within the specified range. If set to **0**, the specified time range is ignored.

- **Use Column Names**: Numerical value (**0** or **1**) that determines whether original column names are retained in the returned table. If **Use Column Names** is set to **1**, original column names are retained. If set to **0**, columns are given generic names. Set this to **0** if you specify multiple groups types.

This function returns a table.

## Group By Time and Unique Values

Returns a table that contains a summary of all the data in a given table, aggregated over both time and index columns. The summary data in the returned table is grouped into specified time intervals, with a further breakdown by unique values in specified index columns.

#### Arguments

The function has the following arguments:

- **Table**: Table whose data is to be summarized.
- **Group Type**: Text string that specifies the type of aggregation to perform. Enter one or more of the following: **sum**, **count**, **average**, **min**, **max**. The default is **sum**. For multiple group types, use a semicolon-separated list and set **Use Column Names** to **0**. The default is **sum**.
- **Date/Time Column Name**: Text string that specifies a column of **Table** that contains a timestamp. If this argument is not supplied, the first column of **Table** is assumed to contain a timestamp.
- **Date Part**: Text string that specifies the date unit to use. Enter **s**, **m**, **h**, **d**, **w**, **M**, **q**, or **y**, for seconds, minutes, hours, days, weeks, months, quarters, or years. The default unit is seconds.
- **Date Parts Per Interval**: Numerical value that specifies he size of each interval in **Date Parts**.
- **Number Of Intervals**: Numerical value that specifies the number of intervals to include in the summary table. The returned table contains one row for each interval. If set to **0**, the number of intervals is determined from the range of data in **Table**, or by the specified time range.
- **Time Range Start**: Text string that specifies the start of the desired time range. If this argument is not supplied, the time range is unbounded at the lower end.
- **Time Range End**: Text string that specifies the end of the desired time range. The time range itself does not include this value, but does include a value that is one second less than **Time Range End**. If this argument is not supplied, the time range is unbounded at the upper end.
- **Restrict To Time Range**: Numerical value (**0** or **1**) that determines whether **Time Range Start** and **Time Range End** are ignored. If set to **1**, the resulting summary table includes

only those time intervals within the specified range. If set to **0**, the specified time range is ignored.

- **Index Column Names**: Text string that specifies the column or columns to be used to form indexes. Separate column names with a semicolon. If this argument is not supplied, the function aggregates only by time interval. If this argument is supplied, the function forms indexes by concatenating the values contained in each index column, and aggregates by index value within time-interval aggregations.
- **Value List**: Table column that contains a set of values to be included in the set of values for the first index column. This is useful if you want the summary table to include values that may or may not be in the **Table** data.
- **Restrict To Value List**: Numerical value (**0** or **1**) that determines whether the table includes only rows that include the values of **Value List**. If set to **1**, only such values are included.
- **Use Column Names**: Numerical value (**0** or **1**) that determines whether original column names are retained in the returned table. If **Use Column Names** is set to **1**, original column names are retained. If set to **0**, columns are given generic names. Set this to **0** if multiple group types are specified.
- **Restrict To Data Combinations**: Numerical value (**0** or **1**) that determines whether the returned table is restricted to only those combinations of values found in the specified index columns that occur in the data. If set to **0**, the returned table contains all possible combinations of unique values found in the specified index columns.

This function returns a table.

## Group by Unique Values

Returns a table that contains a summary of all the data in a given table. The summary data in the returned table is grouped by unique values in specified index columns.

#### Arguments

The function has the following arguments:

- **Table**: Table whose data is to be summarized.
- **Group Type**: Text string that specifies the type of aggregation to perform. Enter one or more of the following: **sum**, **count**, **average**, **min**, **max**. The default is **sum**. For multiple group types, use a semicolon-separated list and set **Use Column Names** to **0**..
- **Index Column Names**: Text string that specifies the column or columns to be used to form indexes. Separate column names with a semicolon. If this argument is not supplied, the function uses the first column of **Table** as the index column. If this argument is supplied, the function forms indexes by concatenating the values contained in each index column, and aggregates by index value.

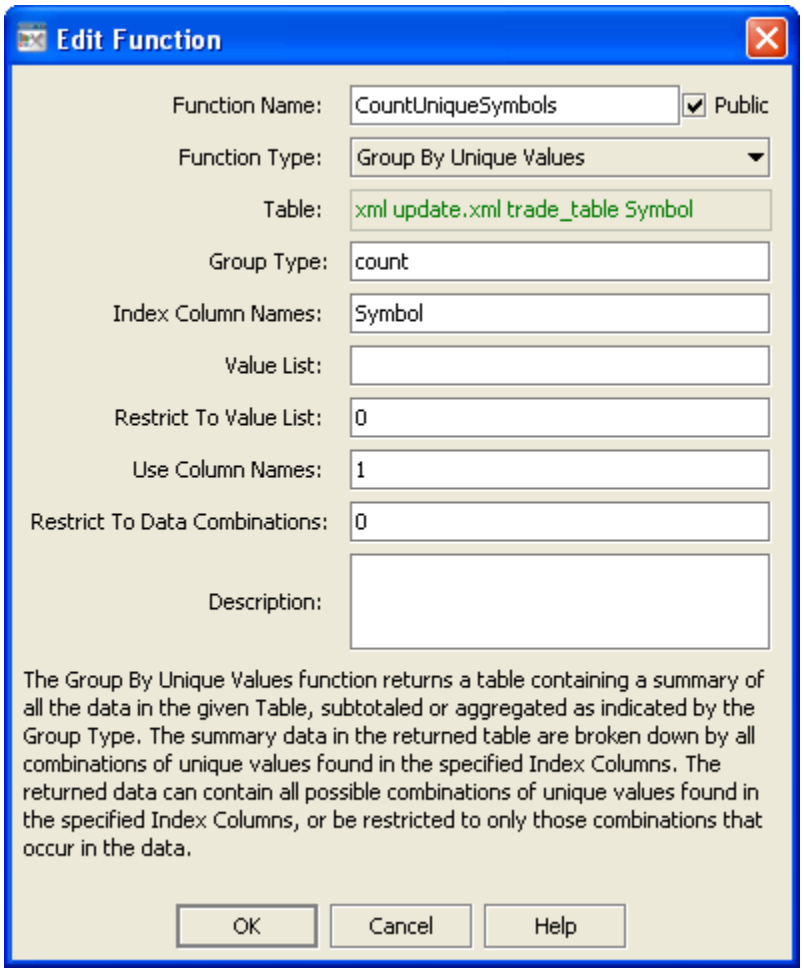
- **Value List**: Table column that contains a set of values to be included in the set of values for the first index column. This is useful if you want the summary table to include values that may or may not be in the **Table** data.
- **Restrict To Value List**: Numerical value (**0** or **1**) that determines whether the table includes only rows that include the values of **Value List**. If set to **1**, only such values are included.
- **Use Column Names**: Numerical value (**0** or **1**) that determines whether original column names are retained in the returned table. If **Use Column Names** is set to **1**, original column names are retained. If set to **0**, columns are given generic names.
- **Restrict To Data Combinations**: Numerical value (**0** or **1**) that determines whether the returned table is restricted to only those combinations of values found in the specified index columns that occur in the data. If set to **0**, the returned table contains all possible combinations of unique values found in the specified index columns. Set this to **0** if multiple group types are specified.

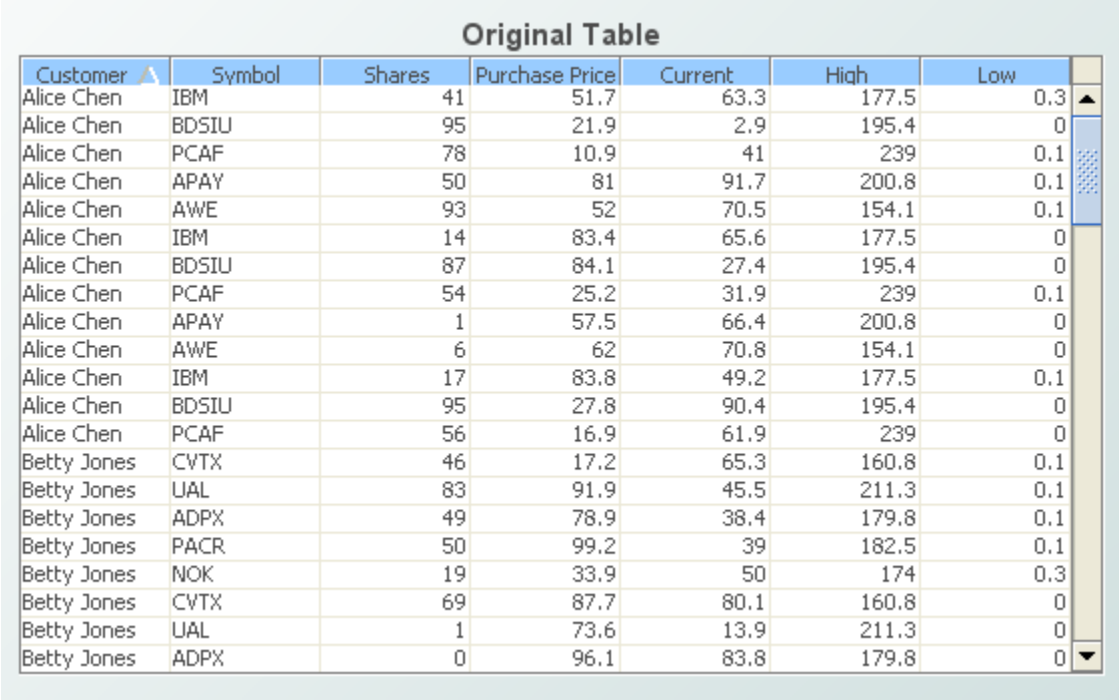
This function returns a table.

#### Examples

The pie graph and the second table below are attached to the function defined by the following dialog. The first table's data table is attached to the argument **Table**.

### Group By Unique Values example 1



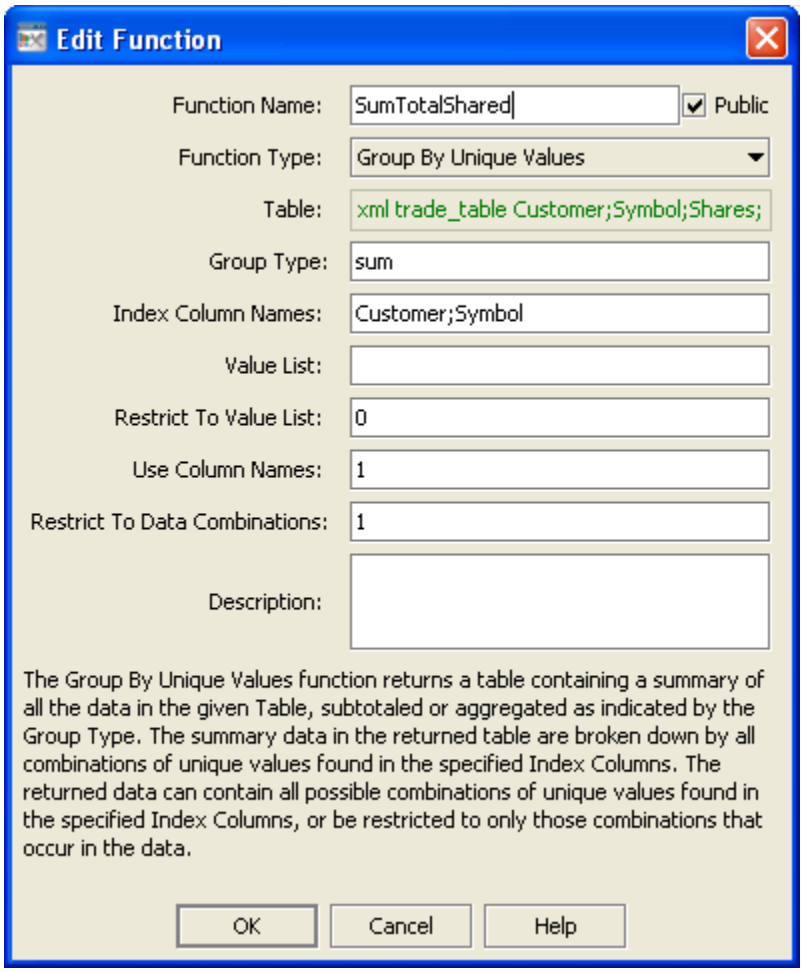


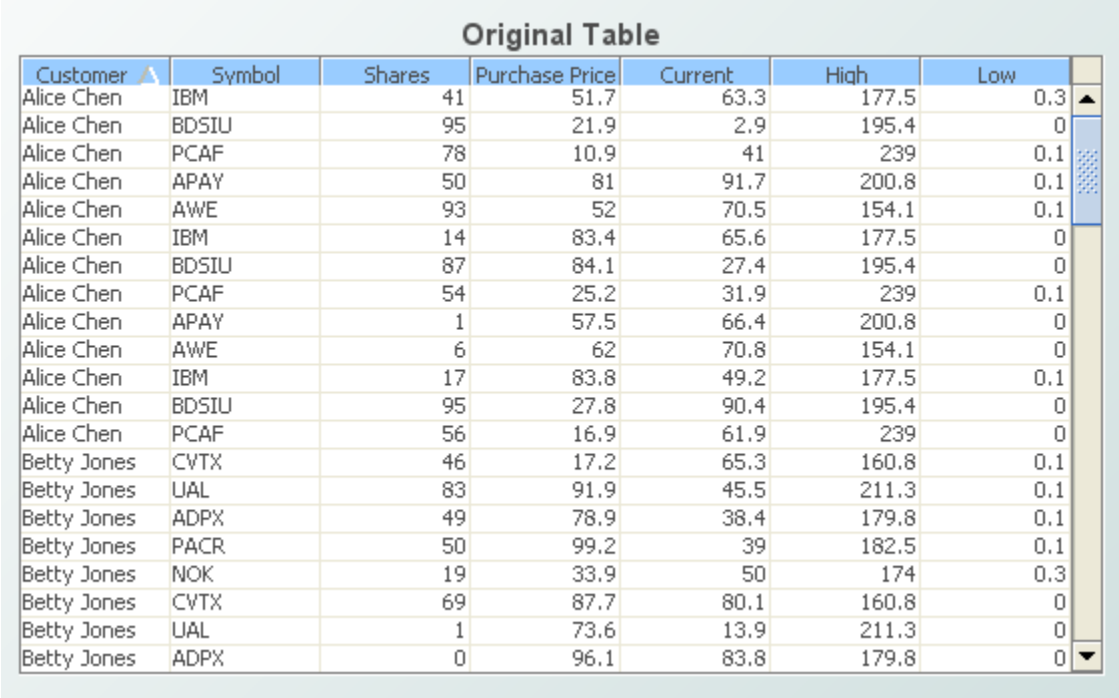
Original Table

| Customer | Symbol | Shares | Purchase Price | Current | High | Low |
|---|---|---|---|---|---|---|
| Alice Chen | IBM | 41 | 51.7 | 63.3 | 177.5 | 0.3 |
| Alice Chen | BDSIU | 95 | 21.9 | 2.9 | 195.4 | 0 |
| Alice Chen | PCAF | 78 | 10.9 | 41 | 239 | 0.1 |
| Alice Chen | APAY | 50 | 81 | 91.7 | 200.8 | 0.1 |
| Alice Chen | AWE | 93 | 52 | 70.5 | 154.1 | 0.1 |
| Alice Chen | IBM | 14 | 83.4 | 65.6 | 177.5 | 0 |
| Alice Chen | BDSIU | 87 | 84.1 | 27.4 | 195.4 | 0 |
| Alice Chen | PCAF | 54 | 25.2 | 31.9 | 239 | 0.1 |
| Alice Chen | APAY | 1 | 57.5 | 66.4 | 200.8 | 0 |
| Alice Chen | AWE | 6 | 62 | 70.8 | 154.1 | 0 |
| Alice Chen | IBM | 17 | 83.8 | 49.2 | 177.5 | 0.1 |
| Alice Chen | BDSIU | 95 | 27.8 | 90.4 | 195.4 | 0 |
| Alice Chen | PCAF | 56 | 16.9 | 61.9 | 239 | 0 |
| Betty Jones | CVTX | 46 | 17.2 | 65.3 | 160.8 | 0.1 |
| Betty Jones | UAL | 83 | 91.9 | 45.5 | 211.3 | 0.1 |
| Betty Jones | ADPX | 49 | 78.9 | 38.4 | 179.8 | 0.1 |
| Betty Jones | PACR | 50 | 99.2 | 39 | 182.5 | 0.1 |
| Betty Jones | NOK | 19 | 33.9 | 50 | 174 | 0.3 |
| Betty Jones | CVTX | 69 | 87.7 | 80.1 | 160.8 | 0 |
| Betty Jones | UAL | 1 | 73.6 | 13.9 | 211.3 | 0 |
| Betty Jones | ADPX | 0 | 96.1 | 83.8 | 179.8 | 0 |

The bar graph and the second table below are attached to the function defined by the following dialog. The first table's data table is attached to the argument **Table**.

### Group By Unique Values example 2



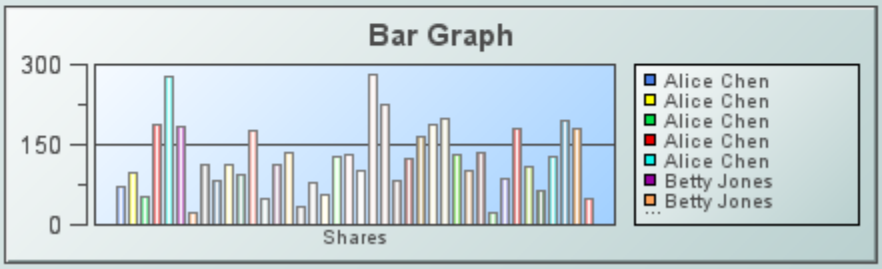


Original Table

| Customer | Symbol | Shares | Purchase Price | Current | High | Low |
|---|---|---|---|---|---|---|
| Alice Chen | IBM | 41 | 51.7 | 63.3 | 177.5 | 0.3 |
| Alice Chen | BDSIU | 95 | 21.9 | 2.9 | 195.4 | 0 |
| Alice Chen | PCAF | 78 | 10.9 | 41 | 239 | 0.1 |
| Alice Chen | APAY | 50 | 81 | 91.7 | 200.8 | 0.1 |
| Alice Chen | AWE | 93 | 52 | 70.5 | 154.1 | 0.1 |
| Alice Chen | IBM | 14 | 83.4 | 65.6 | 177.5 | 0 |
| Alice Chen | BDSIU | 87 | 84.1 | 27.4 | 195.4 | 0 |
| Alice Chen | PCAF | 54 | 25.2 | 31.9 | 239 | 0.1 |
| Alice Chen | APAY | 1 | 57.5 | 66.4 | 200.8 | 0 |
| Alice Chen | AWE | 6 | 62 | 70.8 | 154.1 | 0 |
| Alice Chen | IBM | 17 | 83.8 | 49.2 | 177.5 | 0.1 |
| Alice Chen | BDSIU | 95 | 27.8 | 90.4 | 195.4 | 0 |
| Alice Chen | PCAF | 56 | 16.9 | 61.9 | 239 | 0 |
| Betty Jones | CVTX | 46 | 17.2 | 65.3 | 160.8 | 0.1 |
| Betty Jones | UAL | 83 | 91.9 | 45.5 | 211.3 | 0.1 |
| Betty Jones | ADPX | 49 | 78.9 | 38.4 | 179.8 | 0.1 |
| Betty Jones | PACR | 50 | 99.2 | 39 | 182.5 | 0.1 |
| Betty Jones | NOK | 19 | 33.9 | 50 | 174 | 0.3 |
| Betty Jones | CVTX | 69 | 87.7 | 80.1 | 160.8 | 0 |
| Betty Jones | UAL | 1 | 73.6 | 13.9 | 211.3 | 0 |
| Betty Jones | ADPX | 0 | 96.1 | 83.8 | 179.8 | 0 |

Grouped by Customer and Symbol

| Customer | Symbol | Shares |
|---|---|---|
| Alice Chen | IBM | 72 |
| Alice Chen | AWE | 99 |
| Alice Chen | APAY | 51 |
| Alice Chen | PCAF | 188 |
| Alice Chen | BDSIU | 277 |
| Betty Jones | CVTX | 184 |
| Betty Jones | NOK | 23 |
| Betty Jones | PACR | 112 |
| Betty Jones | ADPX | 82 |
| Betty Jones | UAL | 112 |
| Chris Smith | IBM | 95 |
| Chris Smith | AWE | 177 |




## Join

Returns the result of performing an inner join of two specified tables on specified columns. The result contains all columns from **Left Table** followed by all columns from

**Right Table**, and contains all rows for which the value in **Left Column** exactly matches the value in **Right Column**. **Left Column Name** and **Right Column Name** can each specify a semicolon-separated list of *n* column names, in which case a match occurs if the *i*th value in **Left Column Name** exactly matches the *i*th value in **Right Column Name**, for all *i* between 1 and *n*, inclusive.

### Arguments

The function has the following arguments:

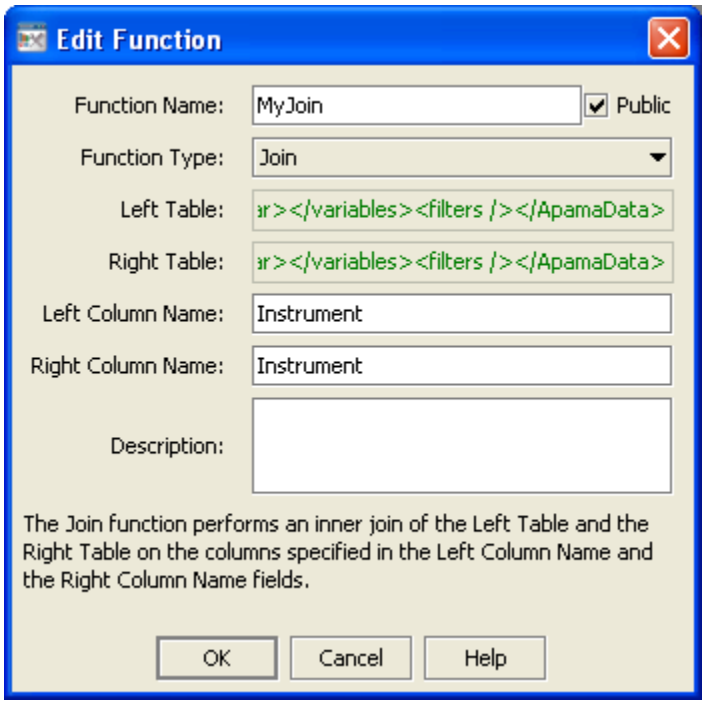
- **Left Table**: Table on which the join is to be performed.
- **Right Table**: Table on which the join is to be performed.
- **Left Column Name**: Text string that specifies the column or columns from **Left Table** on which the join is to be performed. If left blank, the row name, up to the first colon (if it contains a colon), is used instead of a column value.
- **Right Column Name**: Text string that specifies the column or columns from **Right Table** on which the join is to be performed. If left blank, the row name, up to the first colon (if it contains a colon), is used instead of a column value.

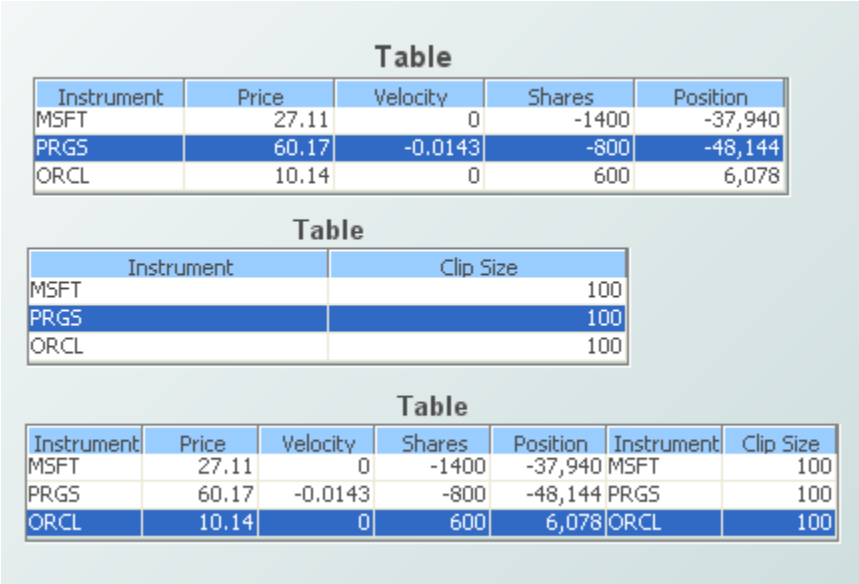
This function returns a table.

### Example

The third table below is attached to the function defined by the following dialog. The first table's data table is attached to the argument **Left Table** and the second table's data table is attached to the argument **Right Table**.

### Join Function example





Table

| Instrument | Price | Velocity | Shares | Position |
|---|---|---|---|---|
| MSFT | 27.11 | 0 | -1400 | -37,940 |
| PRGS | 60.17 | -0.0143 | -800 | -48,144 |
| ORCL | 10.14 | 0 | 600 | 6,078 |

Table

| Instrument | Clip Size |
|---|---|
| MSFT | 100 |
| PRGS | 100 |
| ORCL | 100 |

Table

| Instrument | Price | Velocity | Shares | Position | Instrument | Clip Size |
|---|---|---|---|---|---|---|
| MSFT | 27.11 | 0 | -1400 | -37,940 | MSFT | 100 |
| PRGS | 60.17 | -0.0143 | -800 | -48,144 | PRGS | 100 |
| ORCL | 10.14 | 0 | 600 | 6,078 | ORCL | 100 |

## Join Outer

Performs an outer join of the **Left Table** and the **Right Table** on the columns specified in the **Left Column Name** and the **Right Column Name** fields. The joined table contains all columns from the **Left Table** followed by all columns from the **Right Table**, and contains all rows where the value in the **Left Column** exactly matches the value in the **Right Column**, plus additional rows according to the **Outer Join Type**, which may be **left**, **right**, or **full**.

### Usage notes

In a **left** outer join, the result table includes all the rows from the left table; in a **right** outer join it includes all the rows from the right table, and in a **full** outer join it includes all the rows from both tables. In any row where there is no match for the join column value, the cells from the other table contain null values. (Null values are represented as blank for strings, 0 for integers and longs, `NaN` for floats and doubles, and `NULL_DATE` for dates.)

**Left Column Name** and **Right Column Name** can each specify a semicolon-separated list of *n* column names, in which case a match occurs if the *i*th value in **Left Column Name** exactly matches the *i*th value in **Right Column Name**, for all *i* between 1 and *n*, inclusive.

For a **full** join or **right** join, if the **Left Table** is null, the result is **Right Table**. For a **full** join or **left** join, if **Right Table** is null, the result is **Left Table**. In all other cases the result is null.

### Arguments

The function has the following fields:

- **Left Table**: The first table to be joined.
- **Right Table**: The second table to be joined.
- **Left Column Name**: (Optional) The column or columns in the left table to be joined with the column or columns specified in the **Right Column Name** field. If this field is

left blank, the row name, up to the first : if it contains a :, is used instead of a column value.

- **Right Column Name**: (Optional) The column or columns in the right table to be joined with the column or columns specified in the **Left Column Name** field. If this field is left blank, the row name, up to the first : if it contains a :, is used instead of a column value.
- **Outer Join Type**: Specified as **left**, **right**, or **full**, which may be abbreviated to their first letters. If this field is left blank a full outer join is performed.

This function returns a table.

## Last Table Rows

Returns a table containing one of the following:

- Last *n* rows of a specified table, for a specified number, *n*.
- Last *n* rows for each unique combination of values in a specified set of columns in a specified table, for a specified number, *n*.

#### Arguments

This function has the following arguments:

- **Table**: Table some of whose rows are to be returned.
- **Index Column Names**: Text string that specifies the column or columns to be used to form indexes. Separate column names with a semicolon. If this argument is not supplied, the function returns the last **Number of Rows** of **Table**. If this argument is supplied, the function forms indexes by concatenating the values contained in each index column, and returns **Number of Rows** rows for each index.
- **Number of Rows**: Numerical value that specifies the number rows to be returned, or the number of rows with each index value to be returned.

The function returns a table.

## Mark Time Gaps

Returns a table that results from supplementing a given trend table with special rows that indicate longer-than-expected time intervals between timestamps of adjacent rows of the given trend table. For a trend graph attached to the returned table, the trend line will contain a break (or a specified character) wherever time gaps occurred.

#### Usage Notes

The table is constructed as follows: If the time interval between any two rows in the table is greater than the expected interval, insert 2 new rows between those rows in which the value of each column to be marked is set to NaN or other specified value. (NaN indicates "not a number". A trend graph will break a trace line when a NaN is encountered). The

timestamp of the first new row is set to a value of 1 msec more than the timestamp of the last row before the graph and the timestamp of the second new row is set to a value of 1 msec less than the timestamp of the next row after the gap. It is assumed that the table is sorted by timestamp in ascending order. On the second and subsequent updates of this function, the timestamp of the first row in the table is compared to the timestamp of the last row from the previous update.

#### Arguments

This function has the following arguments:

- **Table**: Table to be checked for time gaps. The table must have a timestamp column and must be sorted by timestamp in ascending order.
- **Name of Timestamp Column**: Name of the table's timestamp column
- **Expected Interval**: The maximum time interval that should occur between consecutive rows in the table. If this interval is exceeded, it is considered a gap. Specify the interval in seconds or specify a value followed by **m**, **h**, **d**, for minutes, hours, or days.
- **Names of Columns to Mark**: Names of the columns to be marked with the specified value when rows are added to mark a gap. Separate multiple column names with semicolons. If no column names are specified then all columns with floating point values will be marked.
- **Mark Columns Width**: The value to be assigned when marking columns in the rows added to mark a gap. The default is NaN, but any numeric value can be used.

The function returns a table.

## Max All Rows or Columns

Determines the maximum cell value for each row or column of the specified Table.

#### Usage notes

If **Return Column** is **1**, the function returns a table with one column. The *n*th cell of the returned column contains the maximum of the numerical cells in the *n*th row of the table specified by the argument **Table**. (If there are no numerical cells in the row, the returned cell contains **0**.)

If **Return Column** is **0**, the function returns a table with one row. The *n*th cell of the returned row contains the maximum of the numerical cells in the *n*th column of the table specified by the argument **Table**. (If there are no numerical cells in the column, the returned cell contains **N/A**, by default—but see the argument **Result Label Column**, below.) The *n*th column of the returned one-row table is labeled with the label of the *n*th column of the table specified by the argument **Table**.

#### Arguments

This function has the following arguments:

- **Table**: Table for which row or column maximums are to be determined.
- **Return Column**: Numeric value that controls whether to return a column or a row of result values. To get a column of result values, one value for each row, set **Return Column** to **1**. To get a row of result values, one value for each column, set **Return Column** to **0**.
- **Result Label**: Text string that specifies a label for the result row or column. If not specified, the label text is **Maximum**. If **Return Column** is **0**, the label appears only if **Result Label Column** is set to a column of **Table** that has no numeric values. If **Return Column** is **1**, the label text always appears and **Result Label Column** is ignored.
- **Result Label Column**: Text string that specifies the column in which **Result Label** appears, if **Return Column** is **0**. The specified column must have no numeric values in order for the label to appear. If **Return Column** is **1**, this argument is ignored.

This function returns a table.

## Max Columns

Returns a table that includes a column reflecting the larger of two specified columns of a specified table, the larger of a specified value and a specified column, or the larger of two specified values.

#### Usage notes

- Case 1: Larger of two specified columns of a specified table. This is the case if **First Column Name or Numeric Value** and **Second Column Name or Numeric Value** both specify a column of **Table**.
- Case 2: Larger of a specified value and a specified column. This is the case if one of **First Column Name or Numeric Value** and **Second Column Name or Numeric Value** specifies a column of **Table** and the other specifies a numeric value.
- Case 3: Larger of two specified values. This is the case if **First Column Name or Numeric Value** and **Second Column Name or Numeric Value** both specify a numeric value.

In case 1, the *n*th cell in the returned column corresponds to the *n*th cell of the table specified by the argument **Table**. Each cell of the returned column contains the larger of the corresponding row's cell in **First Column Name or Numeric Value** and the cell in **Second Column Name or Numeric Value**.

In case 2, the *n*th cell in the returned column corresponds to the *n*th cell of the table specified by the argument **Table**. Each cell of the returned column contains the larger of the corresponding row's cell in the specified column and the specified numeric value.

In case 3, each cell of the returned column contains the larger of the two specified numeric values.

In the returned table, the column that reflects the larger value is preceded by copies of all the columns in **Table**.

#### Arguments

The function has the following arguments:

- **Table**: Table that contains the columns to be compared.
- **First Column Name or Numeric Value**: Text string specifying the first column to be compared, or numerical value to be compared.
- **Second Column Name or Numeric Value**: Text string specifying the second column to be compared, or numerical value to be compared.
- **Result Column Name**: Text string that specifies the name of the column containing the maximums. You must supply a value for this argument.

This function returns a table.

## Min All Rows or Columns

Determines the minimum cell value for each row or column of the specified Table.

#### Usage notes

If **Return Column** is **1**, the function returns a table with one column. The *n*th cell of the returned column contains the minimum of the numerical cells in the *n*th row of the table specified by the argument **Table**. (If there are no numerical cells in the row, the returned cell contains **0**.)

If **Return Column** is **0**, the function returns a table with one row. The *n*th cell of the returned row contains the minimum of the numerical cells in the *n*th column of the table specified by the argument **Table**. (If there are no numerical cells in the column, the returned cell contains **N/A**, by default—but see the argument **Result Label Column**, below.) The *n*th column of the returned one-row table is labeled with the label of the *n*th column of the table specified by the argument **Table**.

#### Arguments

This function has the following arguments:

- **Table**: Table for which row or column minimums are to be determined.
- **Return Column**: Numeric value that controls whether to return a column or a row of result values. To get a column of result values, one value for each row, set **Return Column** to **1**. To get a row of result values, one value for each column, set **Return Column** to **0**.
- **Result Label**: Text string that specifies a label for the result row or column. If not specified, the label text is **Minimum**. If **Return Column** is **0**, the label appears only if **Result Label Column** is set to a column of **Table** that has no numeric values. If **Return Column** is **1**, the label text always appears and **Result Label Column** is ignored.

- **Result Label Column**: Text string that specifies the column in which **Result Label** appears, if **Return Column** is **0**. The specified column must have no numeric values in order for the label to appear. If **Return Column** is **1**, this argument is ignored.

This function returns a table.

## Min Columns

Returns a table that includes a column reflecting one of the smaller of two specified columns of a specified table, the smaller of a specified value and a specified column, or the smaller of two specified values.

#### Usage notes

- Case 1: Smaller of two specified columns of a specified table. This is the case if **First Column Name or Numeric Value** and **Second Column Name or Numeric Value** both specify a column of **Table**.
- Case 2: Smaller of a specified value and a specified column. This is the case if one of **First Column Name or Numeric Value** and **Second Column Name or Numeric Value** specifies a column of **Table** and the other specifies a numeric value.
- Case 3: Smaller of two specified values. This is the case if **First Column Name or Numeric Value** and **Second Column Name or Numeric Value** both specify a numeric value.

In case 1, the *n*th cell in the returned column corresponds to the *n*th cell of the table specified by the argument **Table**. Each cell of the returned column contains the smaller of the corresponding row's cell in **First Column Name or Numeric Value** and the cell in **Second Column Name or Numeric Value**.

In case 2, the *n*th cell in the returned column corresponds to the *n*th cell of the table specified by the argument **Table**. Each cell of the returned column contains the smaller of the corresponding row's cell in the specified column and the specified numeric value.

In case 3, each cell of the returned column contains the smaller of the two specified numeric values.

In the returned table, the column that reflects the smaller value is preceded by copies of all the columns in **Table**.

#### Arguments

The function has the following arguments:

- **Table**: Table that contains the columns to be compared.
- **First Column Name or Numeric Value**: Text string specifying the first column to be compared, or numerical value to be compared.
- **Second Column Name or Numeric Value**: Text string specifying the second column to be compared, or numerical value to be compared.

- **Result Column Name**: Text string that specifies the name of the column containing the minimums. You must supply a value for this argument.

This function returns a table.

## Modulo Columns

Returns a table that includes a column reflecting the remainder of division performed on specified columns of a specified table, the remainder of division performed on a specified value and a specified column, or the remainder of division performed on two specified values.

#### Usage notes

- Case 1: Remainder of division performed on specified columns of a specified table. This is the case if **First Column Name or Numeric Value** and **Second Column Name or Numeric Value** both specify a column of **Table**.
- Case 2: Remainder of division performed on a specified value and a specified column. This is the case if one of **First Column Name or Numeric Value** and **Second Column Name or Numeric Value** specifies a column of **Table** and the other specifies a numeric value.
- Case 3: Remainder of division performed on two specified values. This is the case if **First Column Name or Numeric Value** and **Second Column Name or Numeric Value** both specify a numeric value.

In case 1, the *n*th cell in the returned column corresponds to the *n*th cell of the table specified by the argument **Table**. Each cell of the returned column contains the remainder of dividing the corresponding row's cell in **First Column Name or Numeric Value** by the cell in **Second Column Name or Numeric Value**.

In case 2, the *n*th cell in the returned column corresponds to the *n*th cell of the table specified by the argument **Table**. Each cell of the returned column contains the remainder of dividing the corresponding row's cell in the specified column by the specified numeric value.

In case 3, each cell of the returned column contains the remainder of dividing the first value by the second.

In the returned table, the remainder column is preceded by copies of all the columns in **Table**.

#### Arguments

The function has the following arguments:

- **Table**: Table that contains the columns to be divided.
- **First Column Name or Numeric Value**: Text string specifying the column to whose values are to be used as dividends, or numerical value to be as dividends.

- **Second Column Name or Numeric Value**: Text string specifying the column to whose values are to be used as divisors, or numerical value to be as divisors.
- **Result Column Name**: Text string that specifies the name of the column containing the remainders. You must supply a value for this argument.

This function returns a table.

## Multiply Columns

Returns a table that includes a column reflecting the product of two specified columns of a specified table, the product of a specified value and a specified column, or the product of two specified values.

#### Usage notes

- Case 1: Product of two specified columns of a specified table. This is the case if **First Column Name or Numeric Value** and **Second Column Name or Numeric Value** both specify a column of **Table**.
- Case 2: Product of a specified value and a specified column. This is the case if one of **First Column Name or Numeric Value** and **Second Column Name or Numeric Value** specifies a column of **Table** and the other specifies a numeric value.
- Case 3: Product of two specified values. This is the case if **First Column Name or Numeric Value** and **Second Column Name or Numeric Value** both specify a numeric value.

In case 1, the *n*th cell in the returned column corresponds to the *n*th cell of the table specified by the argument **Table**. Each cell of the returned column contains the result of multiplying the corresponding row's cell in **First Column Name or Numeric Value** by the cell in **Second Column Name or Numeric Value**.

In case 2, the *n*th cell in the returned column corresponds to the *n*th cell of the table specified by the argument **Table**. Each cell of the returned column contains the result of multiplying the corresponding row's cell in the specified column by the specified numeric value.

In case 3, each cell of the returned column contains the product of the two specified numeric values.

In the returned table, the product column is preceded by copies of all the columns in **Table**.

#### Arguments

The function has the following arguments:

- **Table**: Table that contains the columns to be summed.
- **First Column Name or Numeric Value**: Text string specifying the first column to be included in the product, or numerical value to be included in the product.

- **Second Column Name or Numeric Value**: Text string specifying the second column to be included in the product, or numerical value to be included in the product.
- **Result Column Name**: Text string that specifies the name of the column containing the products. You must supply a value for this argument.

This function returns a table.

## Percent Columns

Returns a table that includes a column reflecting the quotient of specified columns of a specified table, the quotient of a specified value and a specified column, or the quotient of two specified values. Values are expressed as percentages.

#### Usage notes

- Case 1: Quotient of specified columns of a specified table, expressed as a percentage. This is the case if **First Column Name or Numeric Value** and **Second Column Name or Numeric Value** both specify a column of **Table**.
- Case 2: Quotient of a specified value and a specified column, expressed as a percentage. This is the case if one of **First Column Name or Numeric Value** and **Second Column Name or Numeric Value** specifies a column of **Table** and the other specifies a numeric value.
- Case 3: Quotient of two specified values, expressed as a percentage. This is the case if **First Column Name or Numeric Value** and **Second Column Name or Numeric Value** both specify a numeric value.

In case 1, the *n*th cell in the returned column corresponds to the *n*th cell of the table specified by the argument **Table**. Each cell of the returned column contains the result, expressed as a percentage, of dividing the corresponding row's cell in **First Column Name or Numeric Value** by the cell in **Second Column Name or Numeric Value**.

In case 2, the *n*th cell in the returned column corresponds to the *n*th cell of the table specified by the argument **Table**. Each cell of the returned column contains the result, expressed as a percentage, of dividing the corresponding row's cell in the specified column by the specified numeric value.

In case 3, each cell of the returned column contains the quotient, expressed as a percentage, of the two specified numeric values.

In the returned table, the percent column is preceded by copies of all the columns in **Table**.

#### Arguments

The function has the following arguments:

- **Table**: Table that contains the columns to be divided.

- **First Column Name or Numeric Value**: Text string specifying the column to whose values are to be used as dividends, or numerical value to be as dividends.
- **Second Column Name or Numeric Value**: Text string specifying the column to whose values are to be used as divisors, or numerical value to be as divisors.
- **Result Column Name**: Text string that specifies the name of the column containing the percentages. You must supply a value for this argument.

This function returns a table.

## Pivot On Unique Values

Returns a table in which row data from a specified table is rotated into columns.

#### Arguments

The function has the following arguments:

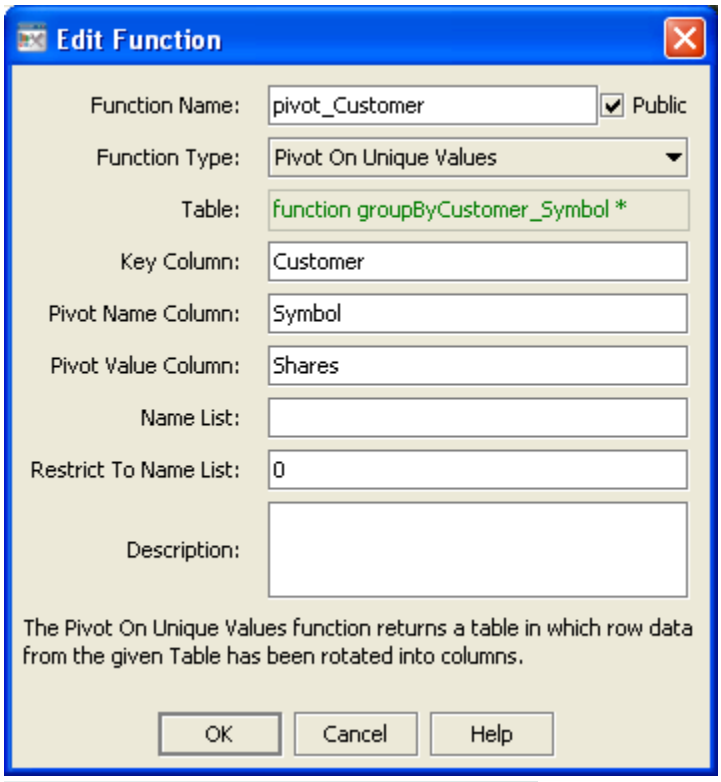
- **Pivot Name Column**: Text string that specifies the column containing values that become new column names in the returned table.
- **Key Column**: Text string that specifies the column used to group rows containing unique names in **Pivot Name Column** into a single row.
- **Pivot Value Column**: Text string that specifies the column containing the data of interest. All consecutive rows that contain the same value in **Key Column** have the data in the **Pivot Value Column** subtotaled into the same row of the resulting table, in the appropriate column.
- **Name List**: Text string that specifies values for which columns should be included in the returned table. To include columns in the returned table for names that are not present in **Pivot Name Column**, specify a semicolon-separated list of names.
- **Restrict to Name List**: Numerical value that determines whether the returned table contains columns *only* for items in **Name List**. If set to 0 or if **Name List** is not specified, all unique values from **Pivot Name Column** are included in the returned table; otherwise only values from the **Name List** are included.

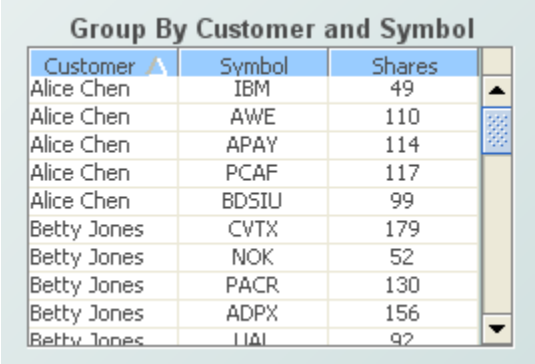
This function returns a table.

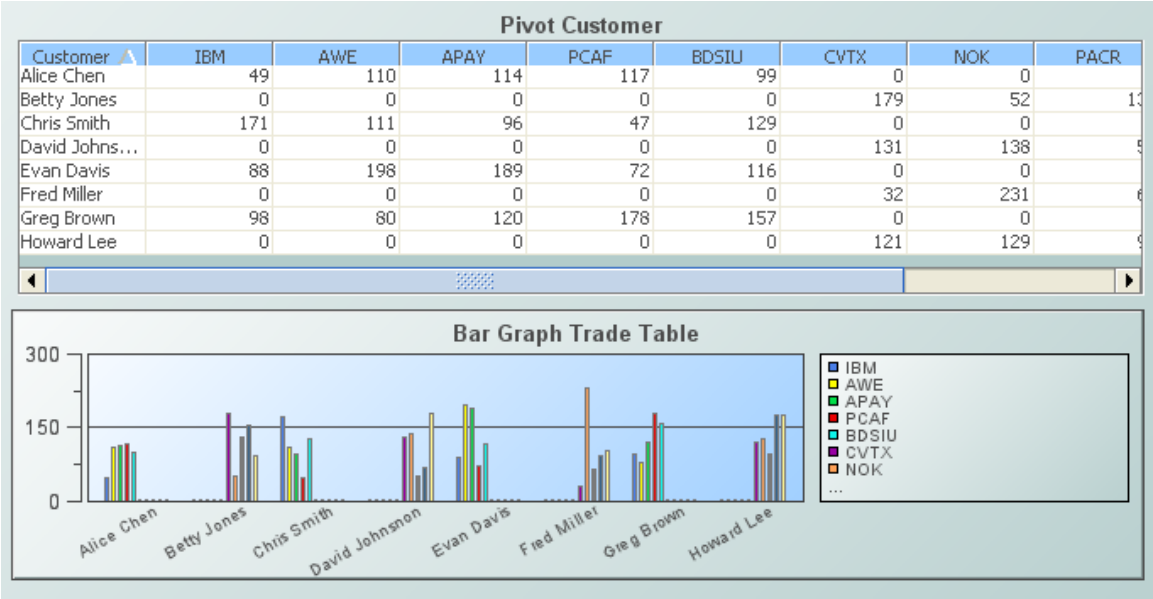
#### Example

The bar chart and the second table (labeled **Pivot Customer**), below, are attached to the function defined by the following dialog. The first table's data table is attached to the argument **Table**.

### Pivot On Unique Values example






## Reference

Returns a reference to the specified table without copying the contents.

### Arguments

The function has the following argument:

- **Table**: The table for which a reference is to be returned.

This function returns a table.

## Rename Columns

Returns a copy of a specified table with specified columns renamed with specified new names.

### Arguments

The function has the following arguments:

- **Table**: The table whose columns are to be renamed.
- **Column Name(s)**: Text string that specifies the columns to be renamed. Supply a single column name or a semicolon-separated list of column names.
- **New Name(s)**: Text string that specifies the new column names. Supply a single name or a semicolon-separated list of names. The *n*th element of **New Name(s)** is the new name of the column specified by the *n*th element of **Column Name(s)**. The number of names in **Column Name(s)** must be less than or equal to the number of names in **New Name(s)**.

This function returns a table.

## Select Column

Returns a one-column table containing only a specified column from a specified table.

#### Arguments

The function has the following arguments:

- **Table**: Table from which the column is to be selected.
- **Select Column Name**: Text string that specifies the name of the column to select.

This function returns a table.

## Set Column Type

Returns a copy of a specified table, with specified columns modified to use specified types.

#### Arguments

The function has the following arguments:

- **Table**: Table from which the column is to be selected.
- **Column Types**: Text string that specifies column-name/type pairs. Separate pairs with spaces. Within each pair, separate column-name from type with a colon, that is, each pair has the following form:
  - *column-name*:*type*

  For *type*, supply one of the following:
  - **STRING**
  - **INTEGER**
  - **LONG**
  - **DOUBLE**
  - **FLOAT**
  - **BOOLEAN**
  - **DATE**

  If *column-name* contains a space or a colon, it must be enclosed in single quotes. Here is an example:
  - **apama.timestamp:DATE 'Max Value':INTEGER Active:BOOLEAN**

This function returns a table.

## Sort Table

Returns a table with the same rows as a specified table but with the rows sorted according to the values in a specified column or columns.

If you specify multiple columns, the returned table is sorted on the first column specified, and then on the second column, and so forth.

#### Arguments

This function has the following arguments:

- **Table**: Table to sort.
- **Sort Column Name**: Text string that specifies the column or columns whose values determine the sort order. Separate column names with a semicolon. If the columns contain text, the sort order is alphabetic, unless the text consists entirely of numbers, in which case the sort order is numeric.
- **Sort Descending**: Numerical value (**0** or **1**) that determines whether the sort order is ascending or descending. If set to **1**, the sort order is descending; otherwise, the sort order is ascending.

**Note:** If an invalid column name is entered, the original table is returned.

This function returns a table.

## Split String

Returns a table with the result of splitting a given string using a specified separator. The returned table contains one column, with a row for each resulting substring.

#### Arguments

The function has the following arguments:

- **String**: Text string to be split.
- **Separator**: Text string consisting of a regular expression that specifies the separator. Use the regular expression form suitable for use with the Java `Pattern` class. See http://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html
- **Results Column Name**: Text string that specifies the name of the returned column.

This function returns a table.

## String to Table

Returns a table whose cell values are specified by a string that uses specified row and column delimiters.

#### Arguments

The function has the following arguments:

- **String**: Text string to be converted into a table.
- **Row Delimiter**: Text string that specifies the delimiter by which the data for one row is separated (in **String**) from the data for the next row.

  Note, if you specify a delimiter that consists of more than one character, those characters are treated as a sequence of delimiters and a new row is created when any one of them is encountered.
- **Column Delimiter**: Text string that specifies the delimiter by which the data for one cell in a row is separated (in **String**) from the data for the next cell in that row. If the table is to contain only one column, do not specify a value for **Column Delimiter**.

  Note, if you specify a delimiter that consists of more than one character, those characters are treated as a sequence of delimiters and a new column is created when any one of them is encountered.
- **Column Name Mode**: Text string that specifies how the function should determine column names for the returned table. Specify one of the following:
  - **AUTO**: Use the generated names **col0**, **col1**, **col2**, and so forth.
  - **STATIC**: Use the names specified in **Column Names**.
  - **STRING**: Use the values specified in the first row of **String**.

  `AUTO` is the default setting; leaving this unset or set to a value other than `AUTO`, `STATIC`, or `STRING` defaults to `AUTO`.
- **Column Names**: Text string that specifies the column names to use in the returned table, if **Column Name Mode** is **Static**.
- **Allow Empty Rows/Columns**: Specifies whether or not empty cells will be created when empty tokens in the string are encountered. Setting this argument to "`true`" or "`1`" means empty cells will be created. For example, in a string that uses "`,`" (comma) as a delimiter, a row represented by `1, , ,4` will result in a row with `1` in the first column followed by two empty cells, followed by `4`. Setting this to "`false`", "`0`", or leaving it unset (the default) means that empty tokens will be ignored. In this case the "`1, , ,4`" example will create a row with `1` in the first column, followed by `4` in the second column, followed by two blank columns.

This function returns a table.

## Subtotal By Time

Returns a table that contains subtotals for the data in a given table. Subtotals are provided for a specified number of specified time intervals.

#### Arguments

The function has the following arguments:

- **Table**: Table for which subtotals are to be provided. The Table must contain a time column and a number column.
- **Date Parts Per Interval**: Numerical value that specifies he size of each interval in **Date Parts**.
- **Number Of Intervals**: Numerical value that specifies the number of intervals to include in the summary table. The returned table contains one row for each interval. If set to **0**, one subtotal row is provided for the entire table.
- **Date Part**: Text string that specifies the date unit to use. Enter **s**, **m**, **h**, **d**, **w**, **M**, **q**, or **y**, for seconds, minutes, hours, days, weeks, months, quarters, or years. The default unit is seconds.
- **Date Format**: Text string that specifies the format of times in the returned table. Specify a pattern string suitable for use with the Java `SimpleDateFormat` class. For example, the format **EEE, hh:mm a** results in a string of the form exemplified by **Wed, 05:32 PM**. Use **q**, **qqq** or **qqqq** for short, medium or long versions of quarter notation. For example, **qqq-yyyy** results in a string of the form exemplified by **Qtr 1-2005**. If no **Date Format** is given, dates are expressed in the form exemplified by **08/30/03 05:32 PM**. If no **Date Format** is given, the type of the first column in the returned table is **Date**; otherwise it is **String**.
- **Use Column Names**: Numerical value (**0** or **1**) that determines whether original column names are retained in the returned table. If **Use Column Names** is set to **1**, original column names are retained. If set to **0**, columns are given generic names. Generic column names are useful when the data attachment for the Table argument uses a substitution that causes the column names to change when the substitution changes.

This function returns a table.

## Subtotal By Unique Values

Returns a table that lists all of the unique values found in the first column of a specified table, along with the sum of the values in the corresponding fields of the remaining columns.

#### Arguments

The function has the following arguments:

- **Table Columns**: Table for which subtotals are to be provided. The function uses the first column of **Table Columns** as the index column. Subtotals are provided for the remaining columns of **Table Columns**.

- **Value List**: Table column that contains a set of values to be included in the set of values for the index column. This is useful if you want the returned table to include values that may or may not be in the **Table Columns** index column.
- **Restrict To Value List**: Numerical value (**0** or **1**) that determines whether the table includes only rows that include the values of **Value List**. If set to **1**, only such values are included.
- **Use Column Names**: Numerical value (**0** or **1**) that determines whether original column names are retained in the returned table. If **Use Column Names** is set to **1**, original column names are retained. If set to **0**, columns are given generic names. Generic column names are useful when the data attachment for the Table argument uses a substitution that causes the column names to change when the substitution changes.

This function returns a table.

## Subtract Columns

Returns a table that includes a column reflecting the difference between two specified columns of a specified table, the difference between a specified value and a specified column, or the difference between two specified values.

#### Usage notes

- Case 1: Difference between two specified columns of a specified table. This is the case if **First Column Name or Numeric Value** and **Second Column Name or Numeric Value** both specify a column of **Table**.
- Case 2: Difference between a specified value and a specified column. This is the case if one of **First Column Name or Numeric Value** and **Second Column Name or Numeric Value** specifies a column of **Table** and the other specifies a numeric value.
- Case 3: Difference between two specified values. This is the case if **First Column Name or Numeric Value** and **Second Column Name or Numeric Value** both specify a numeric value.

In case 1, the *n*th cell in the returned column corresponds to the *n*th cell of the table specified by the argument **Table**. Each cell of the returned column contains the result of subtracting the corresponding row's cell in **Second Column Name or Numeric Value** from the cell in **First Column Name or Numeric Value**.

In case 2, the *n*th cell in the returned column corresponds to the *n*th cell of the table specified by the argument **Table**. Each cell of the returned column contains the result of subtracting the specified numeric value from the corresponding row's cell in the specified column.

In case 3, each cell of the returned column contains the subtracting the second specified value from the first.

In the returned table, the sum column is preceded by copies of all the columns in **Table**.

#### Arguments

The function has the following arguments:

- **Table**: Table that contains the columns to be used in the subtraction operations.
- **First Column Name or Numeric Value**: Text string specifying minuend, the column to be subtracted from, or numerical value to be subtracted from.
- **Second Column Name or Numeric Value**: Text string specifying subtrahend, the column to be subtracted, or numerical value to be subtracted.
- **Result Column Name**: Text string that specifies the name of the column containing the differences. You must supply a value for this argument.

This function returns a table.

## Table Contains Values

Returns a copy of the specified **Table** with a new boolean column containing a value of **true** for each row where the value in the **Comparison Column** is contained in the one and only column of the **Comparison Table**.

This function has the following arguments:

- **Table**: Table to be supplemented with the new boolean-valued column of comparison results.
- **Comparison Column Name**: Name of the column in **Table** whose values are searched for in **Comparison Table**
- **Result Column Name**: Name of the boolean result column to add to **Table**. If no name is specified, the column is named **Result**.
- **Comparison Table**: Single-column table in which to look for values from **Comparison Column** of **Table**

This function returns a table.

# 18 Expression Syntax in Dashboard Functions


- Operators in dashboard function expressions ..... 618
- Arithmetic functions in dashboard function expressions ..... 618
- String functions in dashboard function expressions ..... 621

The following topics describe the syntax you can use when you specify an expression in the following dashboard functions:

- `Evaluate Expression As Double`
- `Evaluate Expression As Row`
- `Evaluate Expression As String`

The syntax for expressions in dashboard functions follows standard Java syntax and includes the operators and functions described in the topics below.

## Operators in dashboard function expressions

A dashboard function that contains an expression can use the following operators:

| Operator | Precedence |
|---|---|
| unary | `+ - !` |
| multiplicative | `* / %` |
| additive | `+ -` |
| relational | `< > <= =>` |
| equality | `== !=` |
| logical | `&& \|\|` |

The following operators are not supported:

- Bitwise `NOT`, `AND`, `OR`, `XOR`
- Arithmetic shift

Supported operators may be applied to `double` type variables. In addition, the relational and equality operators may be applied to `string` variables, and the addition operator may be used to concatenate strings.

## Arithmetic functions in dashboard function expressions

The following arithmetic functions are supported in dashboard function expressions:

- `abs()`

  ```
  double abs(double a)
  ```

  Returns the absolute value of a `double` value.

- `acos()`

  ```
  double acos(double a)
  ```

  Returns the arc cosine of an angle, in the range of 0.0 through pi.

- `asin()`

  ```
  double asin(double a)
  ```

  Returns the arc sine of an angle, in the range of -pi/2 through pi/2.

- `atan()`

  ```
  double atan(double a)
  ```

  Returns the arc tangent of an angle, in the range of -pi/2 through pi/2.

- `atan2()`

  ```
  double atan2(double y, double x)
  ```

  Converts rectangular coordinates (x, y) to polar (r, theta).

- `ceil()`

  ```
  double ceil(double a)
  ```

  Returns the smallest (closest to negative infinity) `double` value that is greater than or equal to the argument and is equal to a mathematical integer.

- `cos()`

  ```
  double cos(double a)
  ```

  Returns the trigonometric cosine of an angle.

- `exp()`

  ```
  double exp(double a)
  ```

  Returns Euler's number e raised to the power of a `double` value.

- `floor()`

  ```
  double floor(double a)
  ```

  Returns the largest (closest to positive infinity) `double` value that is less than or equal to the argument and is equal to a mathematical integer.

- `IEEEremainder()`

  ```
  double IEEEremainder(double f1, double f2)
  ```

  Computes the remainder operation on two arguments as prescribed by the IEEE 754 standard.

- `log()`

  ```
  double log(double a)
  ```

  Returns the natural logarithm (base e) of a `double` value.

- `max()`

  ```
  double max(double a, double b)
  ```

Returns the greater of two `double` values.

- `min()`

```
double min(double a, double b)
```

Returns the smaller of two `double` values.

- `pow()`

```
double pow(double a, double b)
```

Returns the value of the first argument raised to the power of the second argument.

- `random()`

```
double random()
```

Returns a `double` value with a positive sign, greater than or equal to `0.0` and less than `1.0`.

- `rint()`

```
double rint(double a)
```

Returns the `double` value that is closest in value to the argument and is equal to a mathematical integer.

- `round()`

```
long round(double a)
```

Returns the closest `long` value to the argument.

- `sin()`

```
double sin(double a)
```

Returns the trigonometric sine of an angle.

- `sqrt()`

```
double sqrt(double a)
```

Returns the correctly rounded positive square root of a `double` value.

- `tan()`

```
double tan(double a)
```

Returns the trigonometric tangent of an angle.

- `toDegrees()`

```
double toDegrees(double angrad)
```

Converts an angle measured in radians to an approximately equivalent angle measured in degrees.

- `toRadians()`

```
double toRadians(double angdeg)
```

Converts an angle measured in degrees to an approximately equivalent angle measured in radians.

## String functions in dashboard function expressions

The following string functions can be in expressions in dashboard functions:

- `charAt()`

  ```
  char charAt(int index)
  ```

  Returns the `char` value at the specified index.

- `compareTo()`

  ```
  int compareTo(String anotherString)
  ```

  Compares two strings lexicographically.

- `compareToIgnoreCase()`

  ```
  int compareToIgnoreCase(String str)
  ```

  Compares two strings lexicographically, ignoring case differences.

- `concat()`

  ```
  String concat(String value1, String value2)
  ```

  Returns the concatenation of `value1` and `value2`.

- `condExpr()`

  ```
  String condExpr(String expression, String value1, String value2)
  ```

  Evaluated as true or false, returns `value1` if true or `value2` if false.

- `endsWith()`

  ```
  boolean endsWith(String suffix)
  ```

  Tests if this `string` ends with the specified suffix.

- `equals()`

  ```
  boolean equals(Object anObject)
  ```

  Compares this string to the specified object.

- `equalsIgnoreCase()`

  ```
  boolean equalsIgnoreCase(String anotherString)
  ```

  Compares this string to another string, ignoring case considerations.

- `indexOf()`

  ```
  int indexOf(String str, int fromIndex)
  ```

  Returns the index within this string of the first occurrence of the specified substring, starting at the specified index.

- `lastIndexOf()`

  ```
  int lastIndexOf(int ch, int fromIndex)
  ```

Returns the index within this string of the last occurrence of the specified character, searching backward starting at the specified index.

- `length()`

```
int length()
```

Returns the length of this string.

- `replace()`

```
String replace(char oldChar, char newChar)
```

Returns a new string resulting from replacing all occurrences of `oldChar` in this string with `newChar`.

- `startsWith()`

```
boolean startsWith(String prefix, int toffset)
```

Tests if this string starts with the specified prefix beginning at a specified index.

- `substring()`

```
String substring(int beginIndex, int endIndex)
```

Returns a new string that is a substring of this string.

- `toLowerCase()`

```
String toLowerCase()
```

Converts all of the characters in this `String` to lowercase by using the rules of the default locale.

- `toUpperCase()`

```
String to UpperCase()
```

Converts all of the characters in this `String` to uppercase by using the rules of the default locale.

- `trim()`

```
String trim()
```

Returns a copy of the string with leading and trailing whitespace trimmed.

# IV Dashboard Deployment


- Dashboard Deployment Concepts ..... 625
- Generating Dashboards ..... 633
- Preparing Dashboards for Deployment ..... 639
- Deploying Dashboards ..... 649
- Managing the Dashboard Data Server and Display Server ..... 653
- Administering Dashboard Security ..... 677

This section of the documentation provides information and instructions for deploying dashboards. It discusses dashboard deployment considerations, dashboard deployment options, steps for deploying dashboards, tasks for managing Dashboard Data Servers and Dashboard Display Servers, and how to manage dashboard security.

# 19 Dashboard Deployment Concepts


- Deployment options ..... 626
- Data server and display server ..... 628
- Process architecture ..... 628
- Builders and administrators ..... 631

This section discusses fundamental dashboard deployment and administration concepts.

## Deployment options

There are two types of dashboard deployment:

- Web-based: as a simple, thin-client Web page, as an applet, or as a Java Web Start application
- Local: as a locally-installed desktop application (the Dashboard Viewer) together with dashboard-specific files that the application can open

The section on the "Data server and display server" on page 628 discusses some considerations that are relevant to choosing among Web-based deployment options.

### Application installation

Local deployments require the use of the Dashboard Viewer desktop application (available on Windows platforms only). End users open locally-deployed dashboards in the Dashboard Viewer, which must be pre-installed locally or on a shared file system. See the Apama guide *Building and Using Dashboards* for information about using the Dashboard Viewer.

With Web-based deployment, the Dashboard Viewer does not need to be installed locally. Dashboards are invoked through a Web browser, and are installed on demand, as Web pages, applets, or Web Start applications, so they can easily be deployed across a wide area network, including the Internet.

For applet and WebStart deployments, the local Web browser must have the Java plug-in. For simple Web page deployments, no Java plug-in is required. See "Data server and display server" on page 628.

### Authentication

Web-based deployments provide Web-based login functionality and use the authentication mechanism provided by your application server. They support authentication customization by allowing you to, for example, configure your application server to use the security realm and authentication service of your choice.

Local deployments include Data Server login functionality, and support authentication by allowing you to supply any JAAS-supported authentication module as a plug-in to the Data Server and to the Dashboard Viewer.

### Authorization

Web-based deployments support role-based dashboard access control, which allows you to associate a role with a deployed dashboard, and to authorize use of the dashboard only for application-server users with the dashboard's associated role.

Local deployments support dashboard access control by allowing you to use the system security mechanisms in order to restrict access to the deployed dashboard files.

Both types of deployment support Scenario and DataView access control, which allows you to control who can have which type of access to which scenarios and DataViews.

## Data protection

With Web-based deployments you can secure inter-process communication by enabling HTTPS in the application server. With local deployments you can secure inter-process communication by enabling secure sockets (SSL) in the Data Server or Display Server.

With both types of deployment you can secure inter-process communication through the use of secure channels (SSH) and virtual private networks (VPN).

## Refresh latency

The Display Server's minimum refresh latency (5 seconds) is greater than that of the Data Server. Use the Data Server for applications that require high-frequency screen updates.

## Scalability

Both types of deployment are highly scalable, since both use the Data Sever or Display Server to mediate access to Correlators.

## Dashboard support for Apple iPad

Apama dashboards are now supported on Mobile Safari for iOS on the Apple iPad. See the Apama *Supported Platforms* document for details of the versions that are supported.

Any dashboard built with this Apama release is fully functional when viewed on the iPad, provided that the dashboard

- Is a Display Server deployment
- Defines no **System Command** to **Run DOS Command or UNIX Shell**

When you develop dashboards targeted for the iPad, use Layout mode for best results.

End users should be aware of the following characteristics associated with using a dashboard on the iPad:

- If mouseover text and drilldown are both enabled on a visualization object, two touches are required for drilldown. The first touch on an element (for example, a bar of a bargraph) displays the mouseover text for that element, and the second touch on the same element performs the drilldown.
- Drilldown results in a new browser tab, rather than a new window.
- You can scroll pages and listboxes by using two-finger scrolling.
- If you minimize Safari by clicking on the iPad's home button, the display will not update until you wake up the iPad or reopen Safari. In some cases it may be necessary for you to refresh the page from Safari's navigation bar.

## Data server and display server

The Dashboard Data Server and Display Server can each serve as the gateway through which dashboards can access your applications running on the correlator. Use of these Servers provides scalability by obviating client management on the part of the correlator, and provides security by not exposing the correlator directly to clients.

The Data Server mediates correlator access for applet, WebStart, and local deployments; the Display Server mediates correlator access for simple thin-client, Web-page deployments.

The Data Server delivers raw data from which deployed dashboards construct the visualization objects that they display. The Display Server, in contrast, delivers already-constructed visualization objects in the form of image files and image maps, and therefore no Java plug-in is required on clients of the Display Server.

The Display Server's minimum refresh latency (5 seconds) is greater than that of the Data Server. Use the Data Server for applications that require high-frequency screen updates.

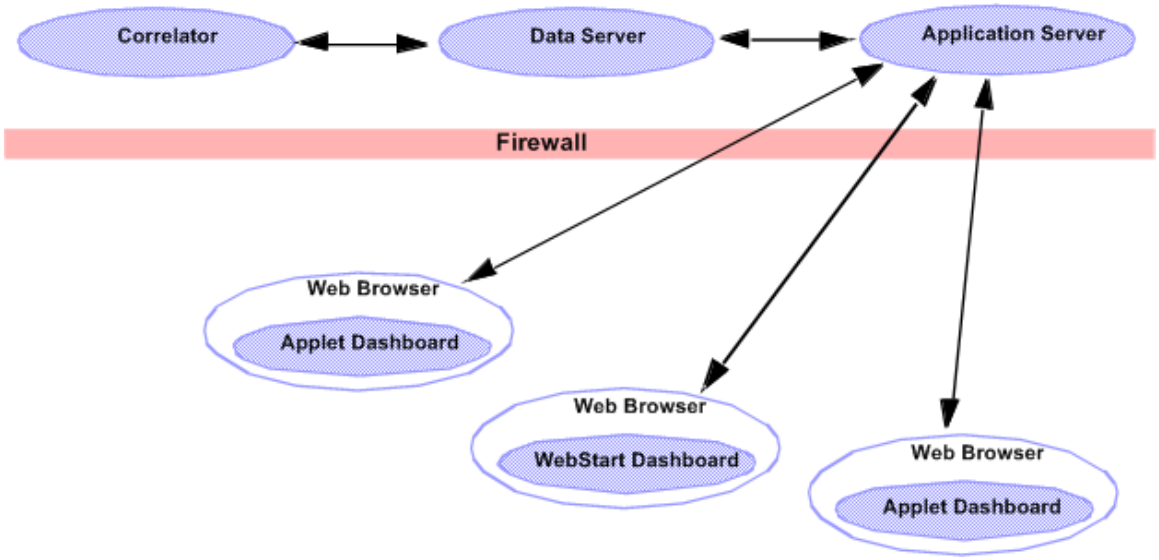
Managing the Data Server and Display Server is covered in "Managing the Dashboard Data Server and Display Server" on page 653.

## Process architecture

Deployed dashboards connect to one or more Correlators via a Dashboard Data Server. As the scenarios in a Correlator run and their variables change, update events are sent to all connected dashboards. When a dashboard receives an update event, it updates its display in real time to show the behavior of the scenarios. User interactions with the dashboard, such as creating an instance of a scenario, result in control events being sent via the Data Server to the Correlator.

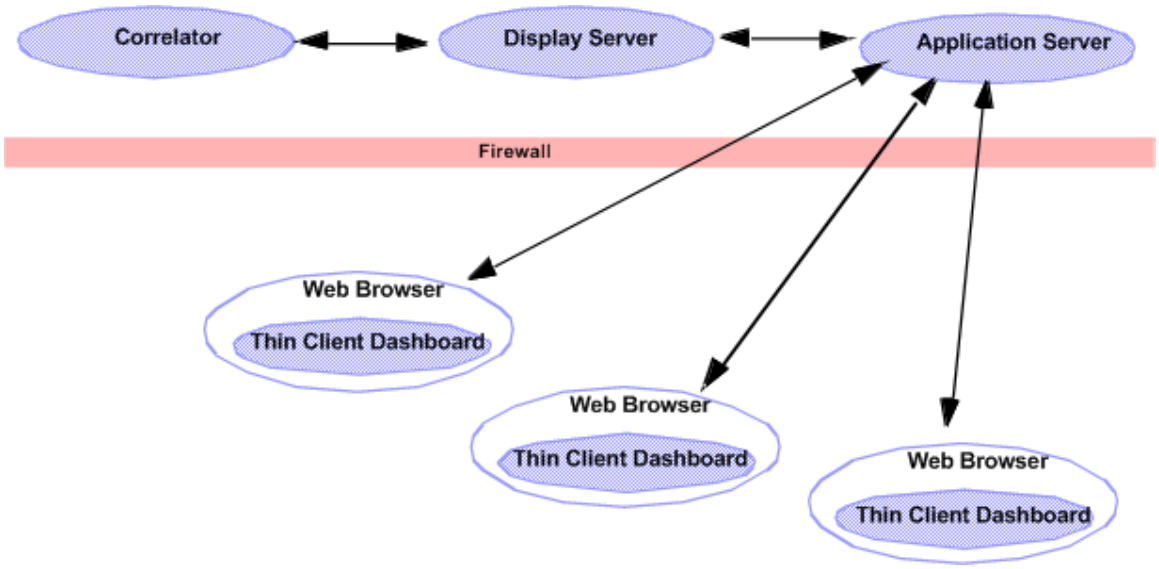
Applet and Web Start dashboards communicate with the Data Server via servlets running on an application server.

The following image shows the process architecture for, applet and WebStart deployments. As you can see, dashboards communicate with your application server, which communicates with the Dashboard Data Server. The Data Server mediates access to the Correlator.



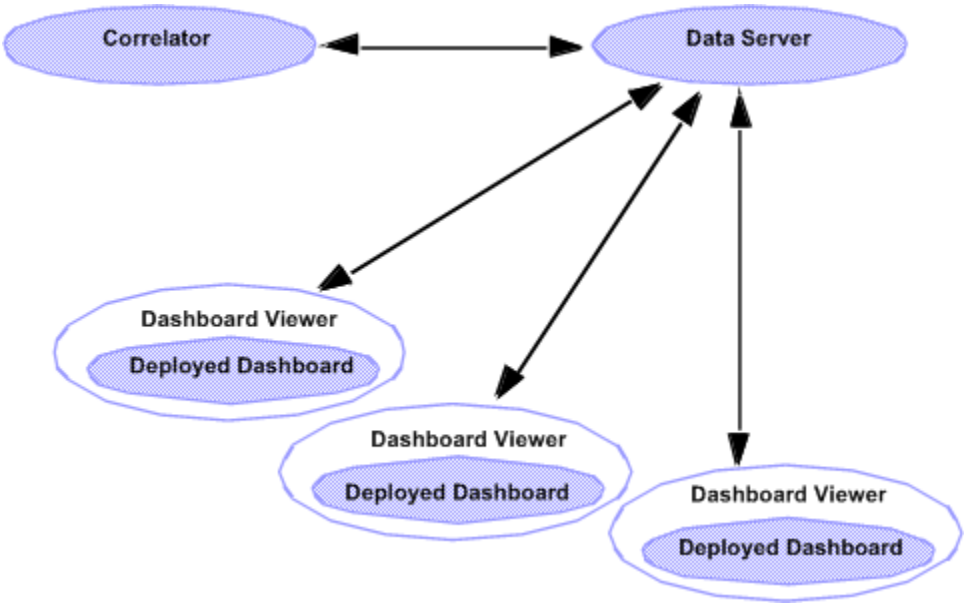

Simple, thin-client, web-page dashboards communicate with the Display Server via servlets that run on your application server. These servlets are bundled with Apama. You must provide your own Java web application server (servlet container). Typically, you install the dashboard servlets in your existing web infrastructure.

The following image shows the process architecture for thin-client, Web-page deployments. Dashboards communicate with your application server, which communicates with the Dashboard Display Server. The Display Server mediates access to the Correlator.



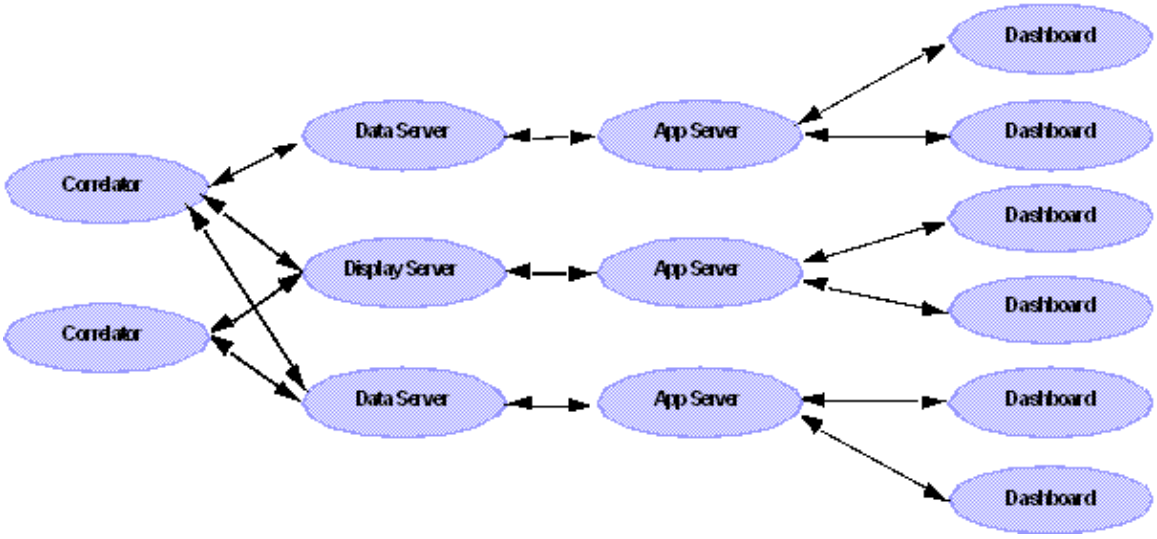

Locally-deployed dashboards communicate directly with the Data Server.

The following image shows the process architecture for local deployments. Dashboards communicate with the Dashboard Data Server, which in turn communicates with the correlator(s).




You can scale your application by adding Data Servers to your configuration. Each Correlator can communicate with multiple Data Servers, and each Data Server can communicate with multiple Correlators.

The following image shows the process architecture after you add Data Servers to your configuration. Each correlator can communicate with multiple Data Servers and Display Servers. Each Data Server and Display Server can communicate with multiple Correlators.




Deployed dashboards have a unique associated default Data Server or Display Server, but advanced users can associate non-default Data Servers with specific attachments and commands. This provides additional scalability by allowing loads to be distributed among multiple servers. This is particularly useful for Display Server deployments. By deploying one or more Data Servers behind a Display Server, the labor of display building can be separated from the labor of data handling. The Display Server can be

dedicated to building displays, while the overhead of data handling is offloaded to Data Servers. See "Working with multiple Data Servers" on page 667 for more information.

## Builders and administrators

There are two types of activity involved in making dashboards available to end users:

- Dashboard development, which requires the use of the Apama Dashboard Builder, as well as the use of the Dashboard Deployment Configuration Editor to generate a deployment package. See "Preparing Dashboards for Deployment" on page 639.
- Dashboard deployment, which requires installing the deployment package, as well as administering the Data Server or Display Server and managing dashboard security.

Sometimes these activities are performed by different individuals. In such a case, the dashboard developer must be sure to communicate the following information to the dashboard administrator regarding the dashboards to be deployed:

- The location and file name of the `.war` file or `.zip` file that was generated by the Deployment Configuration Editor when the developer prepared the dashboard for deployment
- For Display Server deployments, the location of the dashboard project directory (the directory that contains the project's `.rtv` files)
- For Web-based deployments, the Data Sever or Display Server host, port, and update rate that the builder supplied to the Configuration Editor
- The logical name for each correlator as well as the host name and port for each *deployment* correlator (if any) that was specified by the dashboard developer in the **Apama** tab of the **Tools** > **Options...** dialog prior to the generation of the deployment package. See "Changing correlator definitions for deployment" on page 641.
- The trend-data caching requirements for the deployed dashboards. See "Configuring Trend-Data Caching" on page 663.

# 20 Generating Dashboards


- Starting the wizard ..... 634
- Using the wizard ..... 635
- Using the titlebar/toolbar ..... 635
- Using the Introduction form ..... 636
- Using the Main, Create, Edit, and Details Forms ..... 636
- Using the layout configuration forms ..... 637

This section describes how to generate dashboards for a given scenario or query by using the Dashboard Generation wizard.

Dashboards provide the ability to view and interact with scenarios, queries, and DataViews. They contain charts and other objects that dynamically visualize the values of scenario or query variables. Dashboards can also contain control objects for creating, editing, and deleting scenario or query instances. You create Dashboards either with the Dashboard Generation wizard or with the Dashboard Builder.

The wizard allows you to generate simple, default dashboards, customized by your choices regarding layout, visualization objects to display, and scenario or query variables to use with each visualization object.

The Builder is a graphical composition tool that gives you control over a dashboard's appearance and behavior. It also supports a wider array of visualization and control objects than does the wizard. Advanced users can use the Builder instead of the wizard to create dashboards from scratch, or they can use the Builder in conjunction with the wizard to modify or augment generated dashboards. See "Building Dashboard Clients" on page 21 for more information on the Dashboard Builder.

The wizard allows you to:

- Create and edit dashboard-generation configurations
- Save configurations to an XML file
- Generate dashboards from a configuration

Once you have finished generating your dashboards with the wizard (or finished building or modifying them with the Builder), follow the steps described in "Preparing Dashboards for Deployment" on page 639.

Note that if you modify a dashboard with the Builder, the changes you make cannot be propagated back to the configuration that generated the dashboard. So once you modify a dashboard with the Builder, you cannot use the wizard for development of that dashboard.

## Starting the wizard

**To start the wizard**

1. Open your project's `config` folder. If `config` or `config \dashboard_generation.xml` does not exist (because your project was created with a prior Apama release), right click on the project folder and select **Add Dashboard Generation Configuration** from the pop-up menu.
2. Double-click on `dashboard_generation.xml`, or else right click on it and select **Open With > Apama Dashboard Generation Editor** from the pop-up menu. The wizard appears.

## Using the wizard

1. Use the toolbar to create a new dashboard-generation configuration, or to select a previously saved configuration. See "Using the titlebar/toolbar" on page 635.
2. For each of the wizard's forms, type or modify the settings (or accept the default or previously saved settings), and then click **Next**.
3. On the toolbar, click the tool icon in order to generate the dashboards. See "Using the titlebar/toolbar" on page 635.

At any time, you can save configuration changes by selecting **Save** in the Software AG Designer **File** menu.

The wizard's forms are discussed in the topics below.

- "Using the Introduction form" on page 636
- "Using the Main, Create, Edit, and Details Forms" on page 636
- "Using the layout configuration forms" on page 637

## Using the titlebar/toolbar

The titlebar/toolbar is located at the top of the wizard. It allows you to select a configuration to edit. It also allows you to add, remove, and rename configurations.


The titlebar/toolbar includes the following elements:

- Title: At the far left, **Dashboard Generation:** followed by the project name appears.
- **Config** field: A drop down list of configurations appears next to the label **Config:**. These are all the existing configurations for the current project. When you edit a configuration, or generate dashboards for a configuration, you must first select the configuration from the list.
- tool icon: Generates dashboards for the configuration specified in the **Config** field (see above), and displays a dialog that indicates the location of the generated dashboards. Generated dashboards are placed in your project's `dashboards` folder.
- tool icon: Adds a new, named configuration
- tool icon: Renames the configuration specified in the **Config** field (see above).
- tool icon: Removes the configuration specified in the **Config** field (see above).

## Using the Introduction form

The Introduction form appears when the wizard first starts, as well as when you add a new configuration. It allows you to specify a description for the dashboards, a scenario or query for which to generate the dashboards, and a color scheme for the dashboards. It also allows you to specify the types of dashboards to generate.

The **Introduction** form has the following editable components:

- **Description**: Enter an optional text description of the current dashboard-generation configuration.
- **Scenario/Query**: Select the scenario or the query for which the dashboards are to be generated. The visualization objects of the generated dashboards are attached to data from the specified scenario or query variables.

  This drop down menu is disabled if your project has no scenarios or queries. After you import or create a scenario or a query, click on the icon next to the drop down menu to refresh the contents.
- **Scheme**: Select an item from the drop down list in order to control either the dashboard background color or the fill color of visualization objects and forms in the dashboards.
- **Select pages**: Click the checkbox for each type of dashboard that you want to be generated.
- **Main**: First dashboard page that the end user sees in each session. Contains buttons that allow the end user to open other dashboards.
- **Create**: Allows end users to create new scenario or query instances.
- **Edit**: Allows end users to edit the selected scenario or query instance.
- **Details**: Provides a detailed view of scenario or query instances.

## Using the Main, Create, Edit, and Details Forms

You can specify information about each dashboard to be generated (as specified in the **Select pages** section of the **Introduction** form), including height, width, titles, logo, and layout.

One of these forms appears when you click **Next** in the **Introduction** form, and when you click **Next** in the layout configuration form for the previous dashboard page (main, create, or edit).

For each dashboard, the wizard includes a form with the following editable components:

- **Width**: Specify the width of the dashboard window in pixels.
- **Height**: Specify the height of the dashboard window in pixels.

- **Title**: Enter the title of the dashboard window. The title appears in the top, left portion of the window. This field is optional; you can leave it empty.
- **Subtitle**: Enter the subtitle of the dashboard window. The subtitle appears beneath the **Title**. This field is optional; you can leave it empty.
- **Logo**: Select a graphic file. The drop down list includes all supported graphic files in the **dashboard\images** folder under your project folder. Supported formats include `GIF`, `JPG`, and `PNG`.The logo appears in the top, right portion of the generated dashboard. After you import or create a new graphic file, click on the icon next to the drop down menu in order to refresh its contents. This field is optional; you can leave it empty.
- **Layout**: Click the radio button for the desired layout. Each section of a dashboard's layout contains one visualization object (table, bar graph, or pie chart) or one form, as specified in the layout configuration forms.

## Using the layout configuration forms

These forms allow you to enter information about each section of each dashboard's layout, including the visualization object to use in that section, and how to attach the object to scenario or query variables.

For each section of each dashboard (main, create, edit, or details), the wizard includes a form with the following editable elements:

- **Choose content**: Select an object to appear in the current section of the layout. The current section of the layout is indicated by the check mark in the layout diagram to the left of this field.
- Variables table: Use the buttons to ensure that the **Selected variables** column contains those variables that you want attached to the object specified in the **Choose content** field.

  To add variable, select the variable in the **Available variables**, and click **Add**. The variable appears in the **Selected variables** column.

  To remove a variable, select the variable in the **Selected variables** column, and Click **Remove**.

**Summary tables** contain one row for each scenario or query instance, and one column for each variable that is included in the **Selected variables** column. A cell in a given column and row contains the value of the column's corresponding variable for the row's corresponding scenario or query instance. Summary table lists both input and output variables for scenarios and queries.

**Form panels** contain one text-entry field for each scenario or query variable that is included in the **Selected variables** column. Create and Edit dashboards use entered values to initialize or update the variables.

**Bar charts** contain one group of bars for each numeric variable that is included in the **Selected variables** column. Within each group, there is one bar for each scenario instance.

The size of a given bar in a given group is proportional to the value of the group's corresponding numeric variable for the bar's corresponding scenario instance.

**Pie charts** contain one slice for each scenario instance. The size of a given slice is proportional to the value of the first included, numeric variable for the slice's corresponding scenario instance.

**Note:** The layout for pie chart and bar chart does not list any parameters for queries as queries do not have output variables.

If you select **Summary Table** for the **Choose content** field, the following elements are included:

- **Table header**: Enter a label for the table
- **Column name**: Select a variable in the **Selected variables** column of the scenario or query variables table. Enter the header for the selected variable's corresponding table column. Leave this field blank to use the variable name as the column header.
- **Format**: Select a variable in the **Selected variables** column of the scenario or query variables table. Enter a format string for the selected variable's corresponding table column. Specify numerical formats based on the Java format specification, or with the following shorthand:
  - **$** for US dollar money values
  - **$$** for US dollar money values with additional formatting, **( )** for non-money values, formatted similar to money
  - **#** for positive or negative whole values
  - Specify date formats based on the Java date specification.

If you select **Form Panel** for the **Choose content** field, the following element is included:

- **Display name**: Select a variable in the **Selected variables** column of the variables table. Leave the **Display name** field blank to use the variable name as the field label.

If you select **Bar Chart** or **Pie Chart** for the **Choose content** field, the following elements are included:

- **Chart header**: Enter a heading for the chart.
- **Show legend**: Select to show a legend for the chart. The legend indicates the mapping from bar or pie-slice color to value of the scenario's first non-numeric variable for the bar or slice's corresponding scenario instance.
- **Filter by instance**: Select to filter out all scenario instances except the one that corresponds to the selected row in a summary table on the current dashboard. This allows the end user (the user of the generated dashboard) to select the scenario instance to be visualized. In this case, the bar chart has only a single bar for each numeric variable, rather than a group of bars for each numeric variable.

# 21 Preparing Dashboards for Deployment


- Dashboard feature checklist ..... 640
- Changing correlator definitions for deployment ..... 641
- Choosing among deployment types ..... 641
- Using the Deployment Configuration editor ..... 643
- Generating a deployment package from the command line ..... 647
- Sharing information with the Dashboard Administrator ..... 648

This section describes how to prepare a project's dashboards for deployment, including how to create a deployment configuration with the Dashboard Deployment Configuration Editor, as well as how to use the Packaging wizard to generate a deployment package.

Once you have followed the steps described here, if you want to deploy on additional application servers without using Software AG Designer, you or another user must follow the steps described in *Deploying and Managing Apama Applications* in the section *About deploying dashboards*.

**To prepare a project's dashboards for deployment, generate a deployment package**

1. Ensure that the dashboards have the required functionality. See "Dashboard feature checklist" on page 640.
2. Change your dashboard's correlator definitions so that they specify deployment correlators. See "Changing correlator definitions for deployment" on page 641.
3. Decide which type or types of deployment to support for your project. See "Choosing among deployment types" on page 641.
4. Create a deployment configuration or deployment configurations by using the Dashboard Deployment Configuration Editor. See "Using the Deployment Configuration editor" on page 643.
5. Generate a deployment package either with the Dashboard Package wizard or with the `dashboard_management` command line tool. See "Using the Dashboard Package wizard" on page 646 and "Generating a deployment package from the command line" on page 647.
6. If necessary, communicate the appropriate information to the individual who will complete the deployment process. See "Sharing information with the Dashboard Administrator" on page 648.

## Dashboard feature checklist

This section contains a checklist of capabilities that you should include in a project's dashboards in order to ensure that the dashboards provide all standard dashboard functions. Most projects require all these capabilities, but some projects may not.

- Summary view: Displays a listing of all the instances of a scenario or all the items of a DataView.
- Detail view: Provides detailed information about a selected scenario instance or DataView item.
- Create: Allows creation of new scenario instances or DataView items.
- Edit: Supports editing of existing scenario instances or DataView items.
- Delete: Allows deletion of scenario instances or DataView items.

The Statistical Arbitrage sample included with Apama is an example of a scenario dashboard that provides all these capabilities.

## Changing correlator definitions for deployment

When you create a dashboard in Dashboard Builder, use a development correlator. When you deploy a dashboard for use with a live correlator, change the correlator host and port so that they reference the live correlator.

This can be done in two ways:

- In the Dashboard Builder, select **Tools** > **Options...** and use the **Apama** tab to specify the deployment correlator or correlators. You must do this *before* you generate a deployment package with the Dashboard Deployment Configuration Editor. See "Specifying data sources" on page 37.
- When you or another user starts a Data Server or Display Server that will serve event data to your deployed dashboard, use the `-c` or `--correlator` option to override the host and port specified in Dashboard Builder for a given correlator logical name. See "Managing the Dashboard Data Server and Display Server" on page 653.

If a user other than you will complete the deployment, you must communicate to this other user the logical name for each correlator as well as the host name and port for each *deployment* correlator (if any) that you defined.

## Choosing among deployment types

Apama supports two types of dashboard deployment:

- Web-based: as an applet or Java Web Start application, or as a simple, thin-client Web page (thin-client deployment is known as *Display Server deployment*, because it uses the Display Server to mediate correlator access)
- Local: as a locally-installed desktop application (the Dashboard Viewer) together with dashboard-specific files that the application can open

The topics below compare Web-based deployments with local deployments with regard to these factors.

**Note:** The **valueHigh(Low)AlarmCommand** property of Range Dynamic Objects only works for non-display server deployments. For display server deployments, only the **valueHigh(Low)AlarmImage** and the **valueHigh(Low)Color** properties will be honored.

### Application installation

Local deployments require the use of the Dashboard Viewer desktop application (available on Windows platforms only). End users open locally-deployed dashboards in the Dashboard Viewer, which must be pre-installed locally or on a shared file system.

See the Apama guide *Building and Using Dashboards* for information about using the Dashboard Viewer.

With Web-based deployment, the Dashboard Viewer does not need to be installed locally. Dashboards are invoked through a Web browser, and are installed on demand, as Web pages, applets, or Web Start applications, so they can easily be deployed across a wide area network, including the Internet.

For applet and WebStart deployments, the local Web browser must have the Java plug-in. For simple Web page deployments, no Java plug-in is required. See "Data server and display server" on page 628.

## Authentication

Web-based deployments provide Web-based login functionality and use the authentication mechanism provided by your application server. They support authentication customization by allowing you to, for example, configure your application server to use the security realm and authentication service of your choice.

Local deployments include Data Server login functionality, and support authentication by allowing you to supply any JAAS-supported authentication module as a plug-in to the Data Server and to the Dashboard Viewer.

## Authorization

Web-based deployments support role-based dashboard access control, which allows you to associate a role with a deployed dashboard, and to authorize use of the dashboard only for application-server users with the dashboard's associated role.

Local deployments support dashboard access control by allowing you to use the system security mechanisms in order to restrict access to the deployed dashboard files.

Both types of deployment support Scenario and DataView access control, which allows you to control who can have which type of access to which scenarios and DataViews.

## Data protection

With Web-based deployments you can secure inter-process communication by enabling HTTPS in the application server. With local deployments you can secure inter-process communication by enabling secure sockets (SSL) in the Data Server or Display Server.

With both types of deployment you can secure inter-process communication through the use of secure channels (SSH) and virtual private networks (VPN).

## Scalability

Both types of deployment are highly scalable, since both use the Data Sever or Display Server to mediate access to Correlators.

## Choosing among Web-based deployment types

Each Web-based deployment is one of the following:

- Applet
- WebStart application
- Thin-client Web page

The topics below compare these three types of deployment with regard to the following factors.

#### *Installation*

For applet and WebStart deployments, the local Web browser must have the Java plug-in. For thin-client Web page deployments, no Java plug-in is required.

#### *Served data*

The Data Server mediates correlator access for applet, WebStart, and local deployments; the Display Server mediates correlator access for simple thin-client, Web-page deployments.

The Data Server delivers raw data from which deployed dashboards construct the visualization objects that they display. The Display Server, in contrast, delivers already-constructed visualization objects in the form of image files and image maps, and therefore no Java plug-in is required on clients of the Display Server.

#### *Refresh latency*

The Display Server's minimum refresh latency (5 seconds) is greater than that of the Data Server. Use the Data Server for applications that require high-frequency screen updates.

#### *Dashboard command support*

Applet deployments do not support dashboard actions that are system commands (that is, actions that are specified with the Define System Command dialog).

#### *Dashboard iPad Support*

Dashboards targeted for the Apple iPad must be Display Server deployments.

# Using the Deployment Configuration editor

The Deployment Configuration Editor is a form-based interface that allows you to specify dashboard deployment configurations and save them for future use. It also allows you to launch the Packaging wizard, which you can use to generate deployment packages (`.war` files or `.zip` files) from configurations.

## Starting the Configuration editor

**To start the Configuration Editor**

1. If you are using the Project Explorer view, ensure that a project is selected.
2. In either the Project Explorer view or the Workbench Project view, select **New > Dashboard Deployment** from the **File** menu. (You can also right-click in the navigation pane and select **Dashboard Deployment** from the popup menu. In the Workbench Project view, you can also click the **New** button that is above the navigation pane and select **Dashboard Deployment** from the **Apama** folder, or click the down arrow that is next to the **New** button, and select **Dashboard Deployment** from the popup menu.)
3. In the New Dashboard Deployment Configuration dialog, enter a name in the **Configuration** field. The Dashboard Deployment Configuration Editor uses this as the name of the new configuration.
4. Click **Finish**. The Dashboard Deployment Configuration Editor appears, and displays the new dashboard configuration. In addition, a new dashboard-deployment configuration file (`dashboard_deploy.xml`) appears under the current project's **config** folder, if one wasn't already present.

## Saving deployment configurations

Select **Save** in the Software AG Designer **File** menu to save configuration changes.

## Title bar/Toolbar

The title bar/toolbar appears at the top of the Configuration Editor. It allows you to select a configuration to edit. It also allows you to add, remove, and rename configurations, as well as to start the Packaging wizard.

The title bar/toolbar includes the following elements:

- Title: **Dashboard Deploy:** followed by the project name appears at the far left.
- **Configuration** field: A drop down list of configurations appears next to the label **Configuration:**. These are all the existing dashboard configurations for the current project. When you edit a configuration, you must first select the configuration from the list.
-  Dashboard Package: Starts a wizard that can generate deployment packages for one or more of the available configurations.

See "Using the Dashboard Package wizard" on page 646.

-  Add: Adds a new, named configuration
-  Rename: Renames the configuration specified in the **Configurations** field (see above).
-  Remove: Removes the configuration specified in the **Configurations** field (see above).

## General Settings

This section allows you to specify a name, description and deployment type, as well as an entry-point dashboard file or a panels configuration file.

It has the following editable elements:

- **Deploy Name** text field: Enter a name to be used as the file name of the generated deployment package. This name is also used as the directory name for temporary deployment files. Do not use spaces in this field.
- **Choose Deployment Type**: Select the type of deployment for which you want to prepare your dashboard: WebStart, Applet, Display Server, or Local.
- **Dashboard Entry**: Select the dashboard entry point, the file to be used as the initially-displayed dashboard. If you are using multiple display panels, select a panels-initialization file.

If a user other than you will complete the deployment you must communicate to this other user the file name specified in the **Deploy Name** text field as well as the deployment type chosen from the **Choose Deployment Type** drop down list.

## Startup and Server

If you selected a Web-based deployment type (that is, Web Start, Applet, or Display Server deployment), the Startup and Server section is visible.

This section contains the following editable elements:

- **Host** text field: Specify the host of the Data Server or Display Server that will serve data to the deployed dashboard.
- **Port** text field: Specify the port of the Data Server or Display Server that will serve data to the deployed dashboard.
- **Refresh Rate** text field: Specify the dashboard update rate, which is the rate at which the dashboard updates its display to reflect new event data received from the correlator via the Data Server or Display Server. If you know the maximum update rate used by the Servers to which the deployed dashboard might connect, ensure that the update rate that you specify here is no greater than this maximum.

If a user other than you will complete the deployment as described in *Deploying and Managing Apama Applications*, you must communicate to this other user the host, port, and refresh rate that you specified.

## Additional JAR Files

For backward compatibility, this section allows you to specify additional `.jar` files. These additional files must be in the directory `%APAMA_WORK%\dashboards_deploy\lib`.

The JAR files table is for backward compatibility only; specify new `.jar` files in the Software AG Designer **Dashboard Properties** (select **Properties** from the **Project** menu).

Click the **Add** button to display a list of `.jar` files that are found in the directory `%APAMA_WORK%\dashboards_deploy\lib`. Select the file or files that you want to add. To remove files that have been added, select them from the table and click **Remove**.

## Layout

This section allows you to control the appearance of Applet/WebStart deployments. It contains the following editable elements:

- For Applet/WebStart deployments, you can supply a title that will be used by the deployed application or applet. This is optional.
- For Applet deployments, if **Fit Applet To Frame** is checked, the applet's height and width are set to 100%.

## Using the Dashboard Package wizard

The Dashboard Package tool,  (which is on the "Title bar/Toolbar" on page 644) brings up a wizard that guides you through the process of generating a deployment package for specified configurations.

**To use the wizard**

1. Click The Dashboard Package tool  (which is on the "Title bar/Toolbar" on page 644) to start the wizard.

   The **Dashboard Selection** screen appears.

2. In the **Available Configurations** section, use the check boxes to select one or more configurations on which to perform the specified operations and click **Next**. The Package Dashboard Configurations appears.

   The **Package Dashboard Configurations** screen appears.

3. In the **Package Location** field, enter or browse to the path name of the directory into which you want Apama to place the generated deployment package. For local deployments, this is a directory that is accessible to end users.

   If the desired final destination of the deployment package is not accessible to you, the deployment package can be installed by the dashboard administrator as part of the deployment process. See *Deploying Dashboards* in *Deploying and Managing Apama Applications*.

4. Your deployment can include `.jar` files that define custom classes and functions used by your project's dashboards. The `.jar` files that are specified in **Dashboard Properties** (select **Properties** from Software AG Designer's **Project** menu) are automatically included in the generated deployment package. This screen allows you to direct Apama to sign `.jar` files before including them in the deployment package.

Click the **Default** button, to specify the keystore shipped with Apama (`%APAMA_HOME%\etc\DashboardKeystore`). Use the default unless you require a custom keystore. If you require a custom keystore, use the following fields:

- **Signature file** text field: Enter the full path name of (or click Browse... and navigate to) the keystore to use for singing `.jar` files. Leave this field empty to skip signing the `.jar` files.
- **Alias** text field: Enter the private key to be used to sign the `.jar` files. If you are using the keystore shipped with Apama, click the **Default** button (which specifies the alias `dashboard`).
- **Password** text field: Enter the password for the private key specified in the **Alias** text field. Or click the default button for default Alias/Password being used for the DashboardKeystore.

5. Click **Finish**.

   The operations are performed and then the **Dashboard package/deploy/publish summary** appears. The summary indicates which operations succeeded for which configurations. A green check mark indicates success. A red **x** indicates failure.

## Generating a deployment package from the command line

Once you have a defined a dashboard deployment configuration, use `dashboard_management` in order to generate a deployment package. Use the following options:

- `-y` or `--deploy`: Specify a dashboard configuration file, typically `dashboard_deploy.xml` in your project's `config` folder.
- `-c` or `--config`: Specify the name of a deployment configuration that is saved in the file specified with `-y` or `--deploy`.
- `-r` or `--rtvPath`: Specify the directory containing the dashboard (`.rtv` files) to use in order to generate the deployment package.
- `-k` or `--keystoreFile`: Specify the keystore to use in order to sign the `.jar` files to be included in the deployment package. Supply this option only if the `.jar` files are not already signed.
- `-a` or `--alias`: Specify the alias to use in order to sign the `.jar` files to be included in the deployment package.
- `-j` or `--jar`: Specify a third-party jar file to sign. You can specify multiple `-j | --jar` arguments if you have multiple jar files to sign.
- `-o` or `--password`: Specify the password to use in order to sign the `.jar` files to be included in the deployment package.

Here is an example:

```
dashboard_management --deploy "C:\workspace\Demo - Statistical
Arbitrage\config\dashboard_deploy.xml" --config "My Config"
--rtvPath "C:\workspace\Demo - Statistical Arbitrage\dashboards"
```

```
--keystoreFile "%APAMA_HOME%\etc\DashboardKeystore" --alias "dashboard"
--password "terra"
```

For more information on `dashboard_management`, see "Managing and stopping the Data Server and Display Server" on page 672.

## Sharing information with the Dashboard Administrator

There are two types of activity involved in making dashboards available to end users:

- Dashboard development, which requires the use of the Apama Dashboard Builder or Dashboard Generation wizard, as well as the use of the Dashboard Deployment Configuration Editor to generate a deployment package.
- Dashboard deployment, which requires installing and configuring the deployment package, as well as administering the Data Server or Display Server and managing dashboard security.

Sometimes these activities are performed by different individuals. In such a case, the dashboard developer must be sure to communicate the following information to the dashboard administrator regarding the dashboard to be deployed:

- Location and file name of the `.war` file or `.zip` file that was generated by the Deployment Configuration Editor when the developer prepared the dashboard for deployment
- For Display Server deployments, the location of the dashboard project directory (the directory that contains the project's `.rtv` files)
- For Web-based deployments, the Data Sever or Display Server host, port, and update rate that the builder supplied to the Configuration Editor
- Logical name for each correlator as well as the host name and port for each *deployment* correlator (if any) that was specified by the dashboard developer in the **Apama** tab of the **Tools** > **Options...** dialog prior to the generation of the deployment package. See "Changing correlator definitions for deployment" on page 641.
- Trend-data caching requirements for the deployed dashboards. See "Configuring Trend-Data Caching" on page 663.
- Whether SQL-based instance tables are used by the dashboard for data attachments. See "Attaching Dashboards to Correlator Data" on page 55.

# 22 Deploying Dashboards


- Generating the dashboard .war file ..... 650
- Installing a dashboard .war file ..... 650
- Inside a dashboard .war file ..... 651
- Additional steps for display server deployments ..... 651
- Applet and WebStart Deployment ..... 652

Apama dashboards can be deployed to supported application server environments. Use the Dashboard Deployment wizard to generate a `.war` file that you then deploy manually using the deployment tools of your application server. See "Generating Dashboards" on page 633.

There are additional considerations that are covered in the following topics in this section.

If you are a dashboard developer, see also "Preparing Dashboards for Deployment" on page 639.

For a current list of Apama-supported application servers, see Software AG's Knowledge Center in Empower at https://empower.softwareag.com/KnowledgeCenter/default.asp.

## Generating the dashboard .war file

Before deploying a dashboard, you need to first generate the `.war` file for that dashboard. The `.war` file is the deployment package for a dashboard. It contains the webapp, servlets, and supporting resources for deploying a dashboard.

You can generate the dashboard `.war` file from Software AG Designer using Apama's Dashboard Deployment wizard. You can also generate the `.war` file using a command prompt or script with the `dashboard_management` utility.

For more information on generating dashboard `.war` files , see "Preparing Dashboards for Deployment" on page 639.

## Installing a dashboard .war file

When deploying to an application server you need to manually install the dashboard `.war` file to that application server using the tools provided by the server. The details of this vary by application server.

**To install a dashboard .war file**

1. Ensure that you are installing to a supported application server. For an up-to-date listing of supported application servers, see Software AG's Knowledge Center in Empower: (http://empower.softwareag.com).
2. Copy the `.war` file generated by the Dashboard Deployment wizard to the appropriate location for your application server. For example it may have a `webapps` folder.
3. Configure your application server as desired to support this and to secure access to the dashboard as required. The generated `.war` file will have form authentication enabled.
4. Use the deployment tools of your application server to install the dashboard `.war` file.
5. Test that you can access your dashboard and that access is secured as intended.

## Inside a dashboard .war file

A dashboard `.war` file contains the webapp, servlets, and supporting resources necessary to deploy your dashboard and for it to connect to your Apama dashboard data or display server. For the most part you do not need to be aware of the contents of the `.war` file. However, there are several points to consider if you encounter problems.

The generated `.war` file will have form authentication enabled. You must supply the login page for this and configure your application server accordingly. A servlet in the dashboard `.war` file needs the ability to determine the identity of the user displaying a dashboard. This is to enable user based filtering. For this the servlet calls `request.getRemoteUser()`. Any problems calling this will prevent access to the dashboard.

## Additional steps for display server deployments

For thin client deployments you need to provide the Apama dashboard display server with access to the resources used by the dashboard. When the display server builds the displays for users, it requires the `.rtv` files, images, and other files used by the dashboards.

For Display Server deployments, copy your project's dashboard directory (the directory that contains the project's `.rtv` files) to the system on which you want to deploy. By default, the Display Server looks for the deployed project files from its current working directory or from the `APAMA_WORK/dashboards` directory

You need to copy the dashboard project directory, as well as its contents. So, for example, if your dashboard files on the development system are `apama-work/dashboards/MyProject/*.rtv`, they might be located here on the deployment machine:

```
deploy-dir/MyProject/*.rtv
```

In this case, run the Display Server from *`deploy-dir`*, the parent of the dashboard project directory. Do not run the Display Server from the directory `MyProject`.

You can also run the Display Server by using the command line option `--dashboardDir` *`folder`*. If this option is used, then the Display Server does not required to be run from the `APAMA_WORK/dashboards` directory. The deployed project directory described above will be expected under the *`folder`* argument.

Note that the Dashboard Deployment Configuration Editor automatically copies a project's dashboard files to a directory named after the project under the *`APAMA_WORK`*`\dashboards` folder. By default, the `display_server` process will be running in the *`APAMA_WORK`*`\dashboards` directory so the project files will be picked up automatically.

## Applet and WebStart Deployment

WebStart dashboard deployments require that Java WebStart be installed on the end user's local machine. Applet dashboard deployments require that the Java plug-in be installed on the end user's local machine. Browsers block these dashboards by default due to security concerns. To unblock these dashboards, the recommendation is to re-sign the JAR files with a trusted Certificate Authority key. However, there is another way to unblock WebStart and applet dashboards. On each client machine, you can add a server to the **Exception Site List** as described below.

**To add a server to the Exception Site List**

1. Select **Control Panel > Java** to display the **Java Control Panel**.
2. Select the **Security** tab and ensure that the security level is set to **High**.
3. Click **Edit Site List** to display the **Exception Site List** dialog.
4. In the **Exception Site List** dialog, click **Add**.
5. In the **Location** field, enter the dashboard URL. Only the host and port are required. For example:

   ```
   http://MyHost:8080
   ```

6. Click **Add** and then **OK**.
7. Repeat these steps on each client machine. When the dashboard is accessed, a security warning dialog box appears. The risk must be accepted each time in order to view the dashboard.

# 23 Managing the Dashboard Data Server and Display Server


- Prerequisites ..... 654
- Starting and stopping the Data Server or Display Server ..... 654
- Command line options for the Data Server and Display Server ..... 655
- Controlling Update Frequency ..... 662
- Configuring Trend-Data Caching ..... 663
- Managing Connect and Disconnect Notification ..... 667
- Working with multiple Data Servers ..... 667
- Managing and stopping the Data Server and Display Server ..... 672

Use of a deployed dashboard depends on a running Data Server or Display Server. You start, stop, and manage these servers with the following executables, which are found in the `bin` directory of your Apama installation:

- `dashboard_server`
- `display_server`
- `dashboard_management`

On Windows, you can also start the Data Server and Display Server from the **Start** menu. See "Starting and stopping the Data Server or Display Server" on page 654.

## Prerequisites

In order to start a Data Server or Display Server with all the necessary parameters to support a given deployment, you may need to obtain the following information from the Dashboard Builder:

- For Web-based deployments, the Data Sever or Display Server host and port that the builder supplied to the Deployment Configuration Editor when preparing the dashboard for deployment. See "Using the Deployment Configuration editor" on page 643.
- The logical name for each correlator as well as the host name and port for each *deployment* correlator (if any) that was specified by the Dashboard Builder in the **Apama** tab of the **Tools > Options** dialog prior to the generation of the deployment package with the Deployment Configuration Editor. See "Changing correlator definitions for deployment" on page 641.

## Starting and stopping the Data Server or Display Server

Use the following procedure to start and stop the Data Server and Display Server.

**To start and stop the Data Server and Display Server**

- Do one of the following:
  - On Windows, select one of the following commands:
    - **Start > All Programs > Software AG > Start Servers > Start Apama Dashboard Data Server *n.n***
    - **Start > All Programs > Software AG > Start Servers > Start Apama Dashboard Display Server *n.n***
    - **Start > All Programs > Software AG > Stop Servers > Stop Apama Dashboard Data Server *n.n***
    - **Start > All Programs > Software AG > Stop Servers > Stop Apama Dashboard Display Server *n.n***

The current directory for a Display Server that was started from the **Start** menu is the `dashboards` directory of your Apama work directory.

- Use the command line. See "Command line options for the Data Server and Display Server" on page 655.

## Command line options for the Data Server and Display Server

The executable for the Data Server is `dashboard_server` and the executable for the Display Server is `display_server`. They can be found in the `bin` directory of the Apama installation.

### Synopsis

To start the Data Server, run the following command:

```
dashboard_server [ options ]
```

To start the Display Server, run the following command:

```
display_server [ options ]
```

When you run these commands with the `-h` option, the usage message for the corresponding command is shown.

Start the Display Server from the `dashboards` directory of your Apama work directory. To do so, proceed as follows:

1. Open an Apama Command Prompt as described in "Setting up the environment using the Apama Command Prompt" in *Deploying and Managing Apama Applications*.
2. Change to the `%APAMA_WORK%\dashboards` directory.
3. Invoke the executable from the command line.

You can also start Display Server with the `--dashboardDir` *`folder`* option to specify a folder for your deployed dashboards. When `--dashboard` *`folder`* is used, Display Server will be looking for the deployed dashboards from the specified *folder* .

### Description

The `dashboard_server` and `display_server` executables can be run without arguments, in which case they start a server on port `3278` (for a Data Server) or `3279` (for a Display Server) on the local host. Note that these are the default ports used by the Deployment Configuration Editor; see "Using the Deployment Configuration editor" on page 643. You can specify a different port with the `-d` or `--dataPort` option.

The `-c` or `--correlator` option allows you to specify the deployment host and port for a given correlator logical name. See "Changing correlator definitions for deployment" on page 641.

You can enable logging with the `-f` and `-v` (or `--logfile` and `--loglevel`) options or with the `log4j` properties file.

### Options

Both the `dashboard_server` and `display_server` executables take the following options:

| Option | Description |
|---|---|
| `-A \| --sendAllData` | Dashboard data server only. Send all data over the socket regardless of whether or not it has been updated. |
| `-a bool \| --authUsers bool` | Specifies whether to enable user authentication. *`bool`* is one of `true` and `false`. By default, authentication is enabled. Set `--authUsers` to `false` for Web deployments for which authentication is performed by the Web layer. |
| `-c logical-name:host:port:raw-channel \| --correlator logical-name:host:port:raw-channel` | Sets the correlator host and port for a specified logical correlator name. *`raw-channel`* is one of `true` and `false`, and specifies whether to use the raw channel for communication. This overrides the host, port, and raw-channel setting specified by the Dashboard Builder for the given correlator logical name; see "Changing correlator definitions for deployment" on page 641. This option can occur multiple times in a single command. For example:<br>`-c default:localhost:15903:false`<br>`-c work1:somehost:19999:false`<br>These options set the host and port for the logical names `default` and `work1`. |
| `-d port \| --dataPort port` | Data Server or Display Server port to which viewers (for local deployments) or the data servlet (for Web deployments) will connect in order to receive event data. If not specified, the default port (3278 for Data Servers and 3279 for Display Servers) is used. |
| `-E \| --purgeOnEdit bool` | Specifies whether to purge all trend data when a Scenario instance is edited. *bool* is one of `true` and `false`. If this option is not specified, all trend data is purged when an instance is edited. In most cases this is the desired mode of operation. |
| `-F arg \| --filterInstance arg` | Exclude instances which are not owned by user. This option applies to all dashboard processes. |

| Option | Description |
|---|---|
| | Default is `false` for the builder and `true` for the other dashboard processes.<br><br>Exception: when the Dashboard Viewer is connecting to a Dashboard Server, the default is `true` and cannot be overridden. |
| `-f` *`file`* `\|` `--logfile` *`file`* | Full pathname of the file in which to record logging. If this option is not specified, the options in the log4j properties file will be used. |
| `-G` *`file`* `\|` `--trendConfigFile` *`file`* | Trend configuration file for controlling trend-data caching. |
| `-h \| --help` | Emit usage information and then exit. |
| `-J` *`file`* `\|` `--jaasFile` *`file`* | Full pathname of the JAAS initialization file to be used by the Data Server or Display Server. If not specified, the server uses the file `JAAS.ini` in the `lib` directory of your Apama installation. |
| `-L` *`file`* `\|` `--xmlSource` *`file`* | XML data source file. If *`file`* contains static data, append `:0` to the file name. This signals Apama to read the file only once. |
| `-m` *`mode`* `\|` `--connectMode` *`mode`* | Correlator-connect mode. *`mode`* is one of `always` and `asNeeded`. If `always` is specified all correlators are connected to at startup. If `asNeeded` is specified, the Data Server or Display Server connects to correlators as needed. If this option is not specified, the server connects to correlators as needed. |
| `-N` *`name`* `\|` `--name` *`name`* | Component name for identification in correlator |
| `-B` *`logical-name:host:port`* `\|` `--namedServer` *`logical-name:host:port`* | Sets the host and port for a specified logical Data Server name. This overrides the host and port specified by the Dashboard Builder for the given server logical name. This option can occur multiple times in a single command. See "Working with multiple Data Servers" on page 667 for more information. |

| Option | Description |
|---|---|
| `-O file \| --optionsFile file` | Full path of `OPTIONS.ini` |
| `-P n \| --maxPrecision n` | Maximum number of decimal places to use in numerical values displayed by dashboards. Specify values between 0 and 10, or -1 to disable truncation of decimal places. A typical value for `n` is 2 or 4, which eliminates long floating point values (for example, 2.2584435234). Truncation is disabled by default. |
| `-p port \| --port port` | Port on which this Data Server or Display Server will listen for management operations. This is the port used for communication between the server and the `dashboard_management` process. |
| `-Q size \| --queueLimit size` | Set the server output queue size to `size`. This changes the default queue size for each client that is connected to the server. |
| `-q options \| --sql options` | Configures SQL Data Source access. `options` has the following form:<br>`[retry:ms \| fail:n \| db:name \| noinfo \| nopererr \| quote]`<br>`retry`:: Specify the interval (in milliseconds) to retry connecting to a database after an attempt to connect fails. Default is `-1`, which disables this feature.<br>`fail`: Specify the number of consecutive failed SQL queries after which to close this database connection and attempt to reconnect. Default is `-1`, which disables this feature.<br>`db`: Specify the logical name of the database as specified in the builder's SQL options.<br>`noinfo`: Query database for available tables and columns in your database. If a Database Repository file is found, it is used to populate drop down menus in the **Attach to SQL Data** dialog.<br>`nopererr`: SQL errors with the word "permission" in them will not be printed to the console. This is helpful if you have selected the **Use Client Credentials** option for a database. In this case, if your login does |

| Option | Description |
|---|---|
| | not allow access for some data in their display, you will not see any errors.<br><br>`quote`: Encloses all table and column names specified in the **Attach to SQL Data** dialog in quotes when an SQL query is run. This is useful when attaching to databases that support quoted case-sensitive table and column names. If a case-sensitive table or column name is used in the **Filter** field, or you are entering an advanced query in the **SQL Query** field, they must be entered in quotes, even if the `-sqlquote` option is specified. |
| `-R` *`bool`* `\|` `--purgeOnRemove` *`bool`* | Specifies whether to purge all scenario data when an instance is removed. *`bool`* is one of `true` and `false`. If this option is not specified, all scenario data is purged when an instance is removed. |
| `-r` *`bool`* `\|` `--cacheUsers` *`bool`* | Specifies whether to cache and reuse user authorization information. *`bool`* is one of `true` and `false`. Specifying `true` can improve performance, because users are authorized only once (per Data Server or Display Server session) for a particular type of access to particular scenario or scenario Instance. |
| `-s \| --ssl` | Dashboard data server only. Enable secure sockets for client communication. When secure sockets are enabled, the Data Server will encrypt data transmitted to Dashboard Viewers. Encryption is done using the strongest cipher available to both the Data Server and Viewer. SSL certificates are not supported. The Display Server does not support this option. |
| `-T` *`depth`* `\|` `--maxTrend` *`depth`* | Maximum depth for trend data, that is, the maximum number of events in trend tables. If this option is not specified, the maximum trend depth is 1000. Note that the higher you set this value, the more memory the Data Server or Display Server requires, and the more time it requires in order to display trend and stock charts. |

| Option | Description |
|---|---|
| `-t` *`bool`* `\|` `--cacheAuthorizations` *`bool`* | Cache and reuse scenario instance authorizations. Caching authorizations is enabled by default. When caching is enabled, authorization checks are performed only once per user for each scenario or data view they access. Disabling caching allows the current state of the scenario or data view to be used in the authorization check, but can degrade performance. |
| `-u` *`rate`* `\|` `--updateRate` *`rate`* | Data update rate in milliseconds. This is the rate at which the Data Server or Display Server pushes new data to deployed dashboards in order to inform them of new events received from the correlator. *`rate`* should be no lower than 250. If the Dashboard Viewer is utilizing too much CPU, you can lower the update rate by specifying a higher value. If this option is not specified, an update rate of 500 milliseconds is used. |
| `-V \| --version` | Emit program name and version number and then exit. |
| `-v` *`level`* `\|` `--loglevel` *`level`* | Logging verbosity. *`level`* is one of `FATAL`, `ERROR`, `WARN`, `INFO`, `DEBUG`, and `TRACE`. If this option is not specified, the options in the log4j properties file will be used. |
| `-X` *`file`* `\|` `--extensionFile` *`file`* | Full pathname of the extensions initialization file to be used by the Data Server or Display Server. If not specified, the server uses the file `EXTENSIONS.ini` in the `lib` directory of your Apama installation. |
| `-x` *`table-name:key-list`* `\|` `--queryIndex` *`table-name:key-list`* | Add an index for the specified SQL-based instance table with the specified compound key. *`table-name`* is the name of a scenario or DataView. *`key-list`* is a comma-separated list of variable names or field names. If the specified scenario or DataView exists in multiple correlators that are connected to the Dashboard Server, the index is added to each corresponding data table. Example:<br>`--queryIndex Products_Table:prod_id,vend_id`<br>You can only add one index per table, but you can specify this option multiple times in a single command line in order to index multiple tables. |

| Option | Description |
|---|---|
| `-Y \| --enhancedQuery` | Make SQL-based instance tables available as data tables for visualization attachments. See "Attaching Dashboards to Correlator Data" on page 55. |
| `-z` *`zone`* `\| --timezone` *`zone`* | Default time zone for interpreting and displaying dates. *`zone`* is either a Java timezone ID or a custom ID such as `GMT-8:00`. Unrecognized IDs are treated as GMT. See "Timezone ID Values" on page 715 for the complete listing of permissible values for *`zone`*. |
| `--inclusionFilter` *`value`* | Set scenario inclusion filters. Use this option to control scenario/DataView discovery. If not specified, all scenario/DataViews will be discovered and kept in the memory of the dashboard processes, which can be expensive. For example, to include only the `DV_Weather` DataView, specify `--inclusionFilter DV_Weather`. The value can be a comma-separated list of scenario or DataView IDs. If you specify an inclusion filter, any specified exclusion filters are ignored. |
| `--exclusionFilter` *`value`* | Set scenario exclusion filters. Use this option to exclude specific scenarios/DataViews from being kept in the memory of the dashboard processes. If neither exclusion filters nor inclusion filters are specified, all scenario/DataViews will be discovered and kept in the memory of the dashboard processes, which can be expensive. The value can be a comma-separated list of scenario or DataView IDs. If an inclusion filter is specified, any exclusion filters are ignored. |
| `--dashboardDir` *`folder`* | Set the directory where `display_server` will be using to look for the deployed dashboards. If not specified, then `display_server` must be started from `%APAMA_WORK%\dashboards` directory in order for it to locate the deployed dashboards. |
| `--dashboardExtraJars` *`jarFiles`* | A semi-colon separated list of jar files for custom functions, custom commands or any other 3rd party jars (e.g. JDBC jar). If not specified, then the environment variable `APAMA_DASHBOARD_CLASSPATH` must be defined prior to running the dashboard processes. Each entry in *jarFiles* can be an absolute |

| Option | Description |
|---|---|
| | path of the jar file, or when `--dashboardDir` option is used, relative to the *folder* argument. |
| `--namedServerMode` | Dashboard data server only. Specify this option when you start a data server that is used as a named server by a display-server deployment. See "Working with multiple Data Servers" on page 667 for more information. |

## Controlling Update Frequency

The correlator sends update events to the Data Server, Display Server, or any clients using the Scenario Service API, whenever the values of fields or output variables in your DataViews or scenarios change. If you have DataViews or scenarios that update frequently, you might need to reduce the frequency of update events sent by the correlator.

You can adjust settings per scenario definition or globally. The global value is used where a given scenario definition has no specific setting. The per-definition values always take precedence over the global values.

The `com.apama.scenario.ConfigureUpdates` event controls the sending of updates for all Apama-supplied monitors that the ScenarioService can be used with (that is, scenarios as generated by Event Modeler, DataViews, and MemoryStore tables with `exposeMemoryView` or `exposePersistentView` set in their schemas).

A `ConfigureUpdates` event consists of the following:

- `scenarioId` (type `string`): May be the empty string to modify the global values, or a definition's `scenarioId`.
- `Configuration` (type `dictionary (string, string)`): Configuration key and values. Key can be one of:
  - `sendThrottled` (type `boolean`): Whether to send throttled updates (on the `scenarioId.Data` channel). The default is `true`.
  - `sendRaw` (type `boolean`): Whether to send every update (on the `scenarioId.Data.Raw` channel). The default is `true`.
  - `throttlePeriod` (type `float`): Throttle period in seconds. A zero value indicates no throttling. The default is 0.0.
  - `routeUpdates` (type `boolean`): Whether to route `Update` events for the benefit of MonitorScript running in the correlator. The default is `false`.
  - `sendThrottledUser`: (`boolean`): Whether to send throttled updates on a per-user channel. The default is `false`.

- `sendRawUser` (type `boolean`): Whether to send raw updates on a per-user channel. The default is `false`.

Those with a `User` suffix are suitable for using with only custom clients that use `ScenarioServiceConfig.setUsernameFilter()` on their scenario service configuration.

For example, consider the following:

```
com.apama.scenario.ConfigureUpdates("Scenario_scenario1",
    {"sendRaw":"true"})
com.apama.scenario.ConfigureUpdates("",{"sendRaw":"false",
    "throttlePeriod":"0.1"})
com.apama.scenario.ConfigureUpdates("Scenario_scenario2",
    {"sendRaw":"true"})
com.apama.scenario.ConfigureUpdates(Sscenario_scenario3",
    {"throttlePeriod":"1.0"})
```

The above examples configure `Scenario_scenario1` and `Scenario_scenario2` to send raw updates; `Scenario_scenario3` to use a throttle period of 1 second; and all other scenarios to not send raw updates, and to use a throttle period of 0.1 seconds.

Earlier releases used the `com.apama.scenario.SetThrottlingPeriod(x)` event. Note that the use of the `ConfigureUpdates` events allows greater flexibility than the `SetThrottlingPeriod` event (which only controlled sending of throttled updates for all scenarios).

The use of `com.apama.scenario.SetThrottlingPeriod(x)` should be replaced with

```
com.apama.scenario.ConfigureUpdates("", {"throttlePeriod":"x"})
```

Note that by default, `routeUpdates` is false, so any EPL that relies on `Update` (and other scenario control events) to be routed should route a `ConfigureUpdates` event for the `scenarioId`s it is interested in to route `Updates`. Note that scenarios exported as blocks will do this in the generated block code.

The latest values are always used — thus it is not advisable for a client to send an event requesting (for example) raw updates and then undo this when it disconnects, as that will affect other clients. The recommendation is that the administrator should configure settings at initialization time. `routeUpdates` is an exception, but be aware that scenarios for which a block is generated will route updates after a scenario using that block has been injected and deleted, which may not be absolutely required.

Runtime performance of scenarios and dataviews can be improved by setting `sendRaw` and `routeUpdates` to `false` and `throttlePeriod` to a non-zero value. In this case, the cost of an update is reduced (as the `Update` events are only generated when needed, and if throttling, they are only needed at most once every `throttlePeriod`).

## Configuring Trend-Data Caching

By default, dashboard servers (Data Servers and Display Servers) collect trend data for all numeric output variables of scenarios and data views running in their associated correlators. This data is cached in preparation for the possibility that it will be displayed

as historical data in a trend chart when a dashboard starts up. Without the cache, trend charts would initially be empty, with new data points displaying as time elapses.

Advanced users can override the default caching behavior on a given server, and control caching in order to reduce memory consumption on that server, or in order to cache variables that are not cached by default, such as non-numeric variables.

**Important:** In many cases, Server performance can be improved by overriding the default caching behavior, and suppressing the caching of those output variables for which trend-chart historical data is not required.

**Important:** Caching trend data for string variables is very costly in terms of memory consumption.

You control caching with a *trend configuration file*, which allows you to specify the following:

- Individual variables to cache
- Classes of variables to cache
- Default caching rules
- Trend depths (number of data points to maintain) for each scenario and data view

You do not need to provide a trend configuration file. If you provide no trend configuration file, dashboard servers use the default caching behavior described above.

Trend charts can include variables whose trend data is not cached, but they will display no historical (pre-dashboard-startup) data for those variables.

When a Data Server or Display Server starts, it uses the trend configuration file specified with the `-G` option, if supplied. Otherwise it uses the file `trend.xml` in the `dashboards` directory of your Apama work directory, if there is one. (**Note**, Apama provides an example trend configuration file, *`APAMA_HOME`*`\etc\dashboard_onDemandTrend.xml`, that you can copy to *`APAMA_WORK`*`\dashboards\trend.xml` as a basis for a trend configuration file.) Otherwise, it uses the default caching behavior described above.

Here is a sample configuration file:

```
<?xml version="1.0" encoding="UTF-8"?>
<config>
   <trend>
    <item type="SCENARIO" correlator="*" name="*"
      vars="ALL_NUMERIC_OUTPUT" depth="10000"/>
     <item type="SCENARIO" correlator="*" name="Scenario_scenario1"
      vars="LIST" depth="5000">
        <var name="A"/>
        <var name="B"/>
    </item>
    <item type="DATAVIEW" correlator="production" name="DV_dataview1"
      vars="LIST" depth="5000">
        <var name="A"/>
        <var name="B"/>
    </item>
  </trend>
 </config>
```

This file specifies the following:

- For `Scenario_scenario1` in all correlators, cache trend data for variables `A` and `B` with a maximum trend depth of 5000.
- For all other scenarios, cache all numeric output variables with a maximum trend depth of 10,000.
- For `DV_dataview1` in correlator `production`, cache variables `A` and `B` with a maximum trend depth of 5000.
- For all other data views, cache no trend data.

In general, a trend configuration file is an XML file that includes of one or more `item` elements with the following attributes:

- `type`:`SCENARIO` or `DATAVIEW`
- `correlator`: Logical name of correlator. Use `*` for if the item applies to all correlators
- `name`: Scenario or data view ID. Use `*` if the item applies to all scenarios or data views.
- `vars`: Class of variables to cache trend data for. Specify one of the following:
  - `LIST`: Cache the individual variables that are listed in `var` sub-elements.
  - `ALL`: Cache all input and output variables.
  - `ALL_OUTPUT`: Cache all output variables.
  - `ALL_NUMERIC_OUTPUT`: Cache all numeric output variables.
- `depth`: Maximum depth of trend data to cache.

If the `vars` attribute of an item element is `LIST`, the element has zero or more `var` sub-elements. Each `var` element has single attribute, `name`, which specifies the name of a scenario variable or data view field.

The `item` elements are nested in a `trend` element, which is nested within a `config` element.

If a particular data view or scenario on a given correlator matches multiple `item` elements in a server's trend configuration file, the server chooses the *best-matching*`item` and caches the variables specified in that `item`. Following are the ways, in order from best to worst, in which an `item` can match a data view or scenario on a given correlator:

1. Fully resolved: Exact match for both correlator name and scenario or data-view name
2. Wildcard correlator: Wildcard correlator and exact match for scenario or data-view name
3. Wildcard scenario: Exact match for correlator name and wildcard scenario or data-view

4. Fully wildcarded: Wildcard correlator and wildcard scenario or data view

If there are multiple best matches, the last match is used.

Consider, for example, scenarios named `Scenario_scenario1` and `Scenario_scenario2`, correlators named `production` and `development`, and the following `item` elements:

```
1 <item type="SCENARIO" correlator="production" name="Scenario_scenario1"
    vars="LIST" depth="5000">
2 <item type="SCENARIO" correlator="*" name="Scenario_scenario1" vars="LIST"
    depth="5000">
3 <item type="SCENARIO" correlator="production" name="*" vars="LIST"
    depth="5000">
4 <item type="SCENARIO" correlator="*" name="*" vars="LIST"
    depth="5000">
```

`Scenario_scenario1` running on `production` best matches `item`*1*.

`Scenario_scenario1` on `development` best matches `item`*2*.

`Scenario_scenario2` on `production` best matches `item`*3*.

`Scenario_scenario2` on `development` best matches `item`*4*.

Below are some additional sample configuration files. The following file caches trend data for all input and output variables:

```
<?xml version="1.0" encoding="UTF-8"?>
<config>
   <trend>
    <item type="SCENARIO" correlator="*" name="*" vars="ALL"
      depth="10000"/>
    <item type="DATAVIEW" correlator="*" name="*" vars="ALL"
      depth="10000"/>
  </trend>
 </config>
```

The following caches trend data for all numeric output variables, the default behavior:

```
<?xml version="1.0" encoding="UTF-8"?>
<config>
   <trend>
    <item type="SCENARIO" correlator="*" name="*"
      vars="ALL_NUMERIC_OUTPUT" depth="10000"/>
    <item type="DATAVIEW" correlator="*" name="*"
      vars="ALL_NUMERIC_OUTPUT" depth="10000"/>
  </trend>
 </config>
```

The following caches no data, which results in trend-data collection only on demand:

```
<?xml version="1.0" encoding="UTF-8"?>
<config>
   <trend>
  </trend>
 </config>
```

The following caches a single variable for a single scenario:

```
<?xml version="1.0" encoding="UTF-8"?>
<config>
   <trend>
    <item type="SCENARIO" correlator="*" name="Scenario_scenario1"
```

```
      vars="LIST" depth="5000">
        <var name="PRICE"/>
    </item>
  </trend>
 </config>
```

## Managing Connect and Disconnect Notification

Whenever a dashboard connects to or disconnects from a Data Server or Display Server, the server sends a special notification event to all connected correlators that include the Dashboard Support bundle.

The events are defined as follows:

```
event DashboardClientConnected {
  string userName;
  string sessionId;
  dictionary<string,string> extraParams;
}event DashboardClientDisconnected {
  string userName;
  string sessionId;
  dictionary<string,string> extraParams;
}
```

`userName` specifies the user name with which the dashboard was logged in to the server.

`sessionID` is a unique identifier for the dashboard's session with the server.

`extraParams` may be used in a future release.

Note that the circumstances under which a dashboard disconnects from a server include but are not limited to the following:

- End user exits the Dashboard Viewer or Web browser in which a dashboard is loaded.
- End user exits a Web browser tab in which a dashboard is loaded.
- Network failure causes loss of connectivity to Viewer or Web browser in which a dashboard is loaded.

Note also that disconnect notification might be sent only after a timeout period rather than immediately upon loss of connection.

Follow these steps to manage connect and disconnect notification:

1. Ensure that the Dashboard Support bundle is loaded into all relevant correlators.
2. Use scenarios or monitors to process `DashboardClientConnected` and `DashboardClientDisconnected` events. Base processing on the values of the `userName` and `sessionId` fields.

## Working with multiple Data Servers

Deployed dashboards have a unique associated default Data Server or Display Server. For Web-based deployments, this default is specified in the Startup and Server section of the Deployment Configuration Editor. For Viewer deployments, it is specified upon

Viewer startup. By default, the data-handling involved in attachments and commands is handled by the default server, but advanced users can associate non-default Data Servers with specific attachments and commands. This provides additional scalability by allowing loads to be distributed among multiple servers. This is particularly useful for Display Server deployments. By deploying one or more Data Servers behind a Display Server, the labor of display building can be separated from the labor of data handling. The Display Server can be dedicated to building displays, while the overhead of data handling is offloaded to Data Servers.

Apama supports the following multiserver configurations:

- Builder with multiple Data Servers. See "Builder with multiple Data Servers" on page 668.
- Viewer with multiple Data Servers. See "Viewer with multiple Data Servers" on page 669.
- Display Server (thin client) deployment with multiple Data Servers. See "Display Server deployments with multiple Data Servers" on page 670.
- Applet or WebStart deployment with multiple Data Servers. See "Applet and WebStart deployments with multiple Data Servers" on page 671.

The **Attach to Apama** and **Define ... Command** dialogs (except **Define System Command**) include a **Data Server** field that can be set to a Data Server's logical name. To associate a logical name with the Data Server at a given host and port, developers use the **Data Server** tab in the **General** tab group of the **Application Options** dialog (select **Tools Options** in Builder).

For Display Server (thin client) deployments, you must use the option `--namedServerMode` whenever you start named Data Servers. See "Display Server deployments with multiple Data Servers" on page 670.

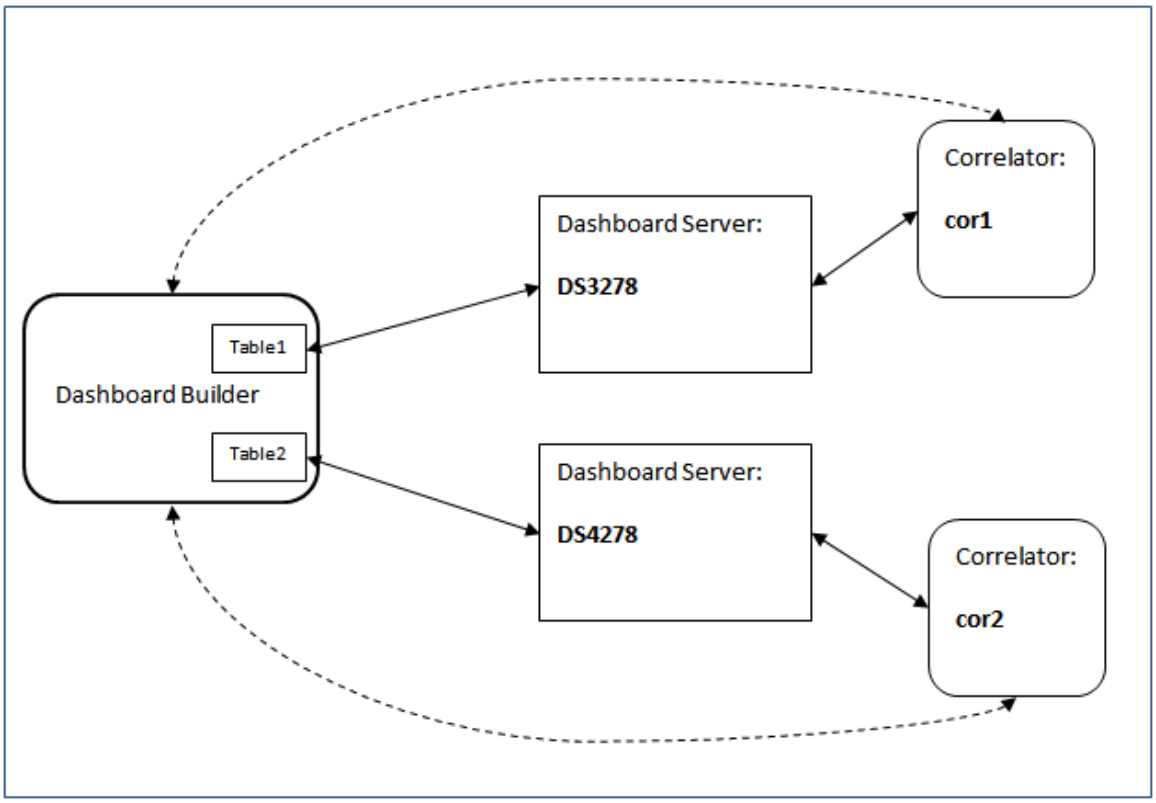
The logical Data Server names specified in the Builder's **Application Options** dialog are recorded in the file `OPTIONS.ini`, and the deployment wizard incorporates this information into deployments. You can override these logical name definitions with the `--namedServer` *name*`:`*host*`:`*port* option to the Builder, Viewer, Data Server or Display Server executable. Below is an example. This is a sequence of command line options which should appear on a single line as part of the command to start the executable:

```
--namedServer Server1:ProductionHost_A:3278 --
namedServer Server2:ProductionHost_B:4278 --namedServer
Server3:ProductionHost_C:5278
```

Here `Server1`, `Server2` and `Server3` are the server logical names.

## Builder with multiple Data Servers

Builder maintains connections with the Data Servers named in attachments and commands. Note that it connects directly to the correlator (dotted lines in the figure below) in order to populate dialogs with metadata. Correlator event data is handled by the Data Servers.



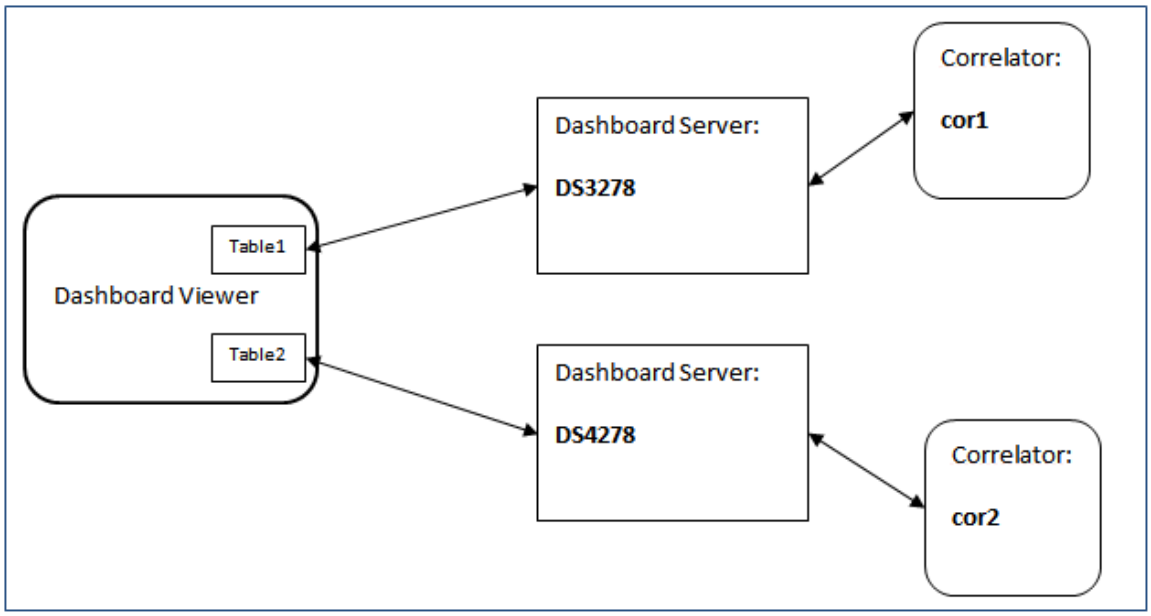

You can override the logical server names specified in the **Application Options** dialog with the `--namedServer` *name* `:`*host* `:`*port* option to the Builder executable. Below is an example. This is a sequence of command line options which should appear on a single line as part of the command to start the executable:

```
--namedServer Server1:ProductionHost_A:3278 --
namedServer Server2:ProductionHost_B:4278 --namedServer
Server3:ProductionHost_C:5278
```

Here `Server1`, `Server2` and `Server3` are the server logical names.

## Viewer with multiple Data Servers

Viewer maintains connections with the Data Servers named in attachments and commands of opened dashboards.




In the Data Server Login dialog (which appears upon Viewer startup), end users enter the host and port of the default Data Server (or accept the default field values). If all attachments and commands use named Data Servers, end users can check the **Only using named data server connections** check box and omit specification of a default server.

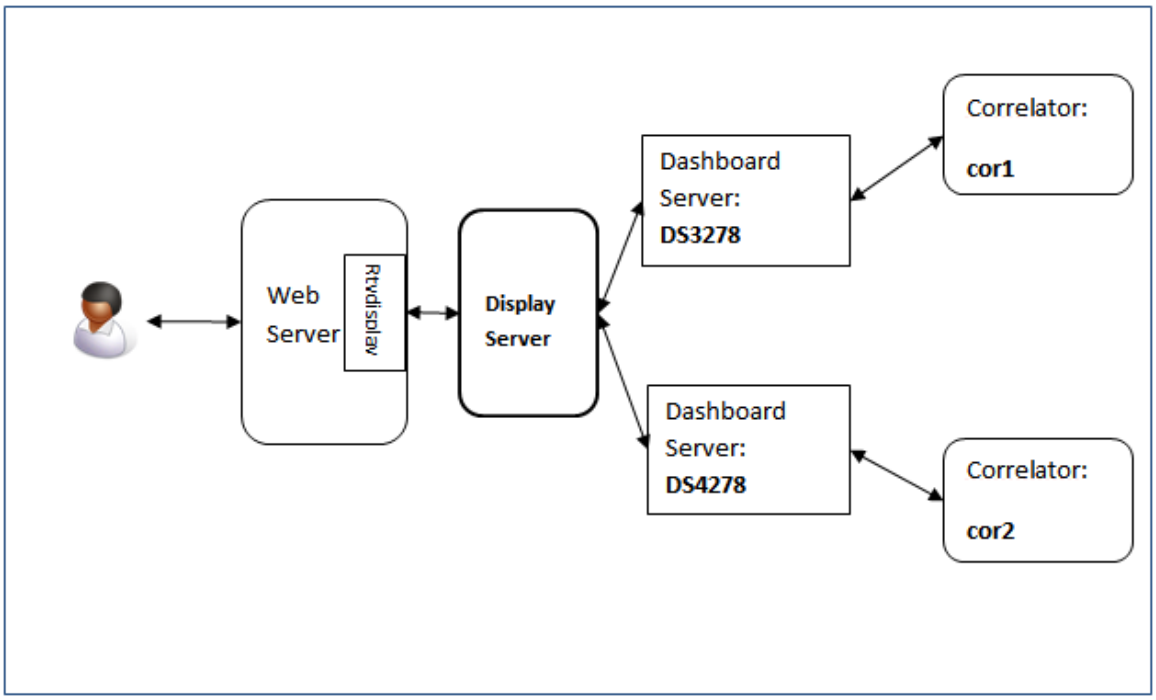
The logical data server names specified in the Builder's **Application Options** dialog are recorded in the file `OPTIONS.ini`, which is found in the deployed `.war` file along with dashboard `.rtv` files. You can override these logical name definitions with the `--namedServer` *name*`:`*host*`:`*port* option to the Viewer executable. Below is an example. This is a sequence of command line options which should appear on a single line as part of the command to start the executable:

```
--namedServer Server1:ProductionHost_A:3278 --
namedServer Server2:ProductionHost_B:4278 --namedServer
Server3:ProductionHost_C:5278
```

Here `Server1`, `Server2` and `Server3` are the server logical names.

## Display Server deployments with multiple Data Servers

The Display Server maintains connections with the Data Servers named in attachments and commands of its client dashboards.




In a Display Server deployment, each named Data Server must be started with the `--namedServerMode` option.

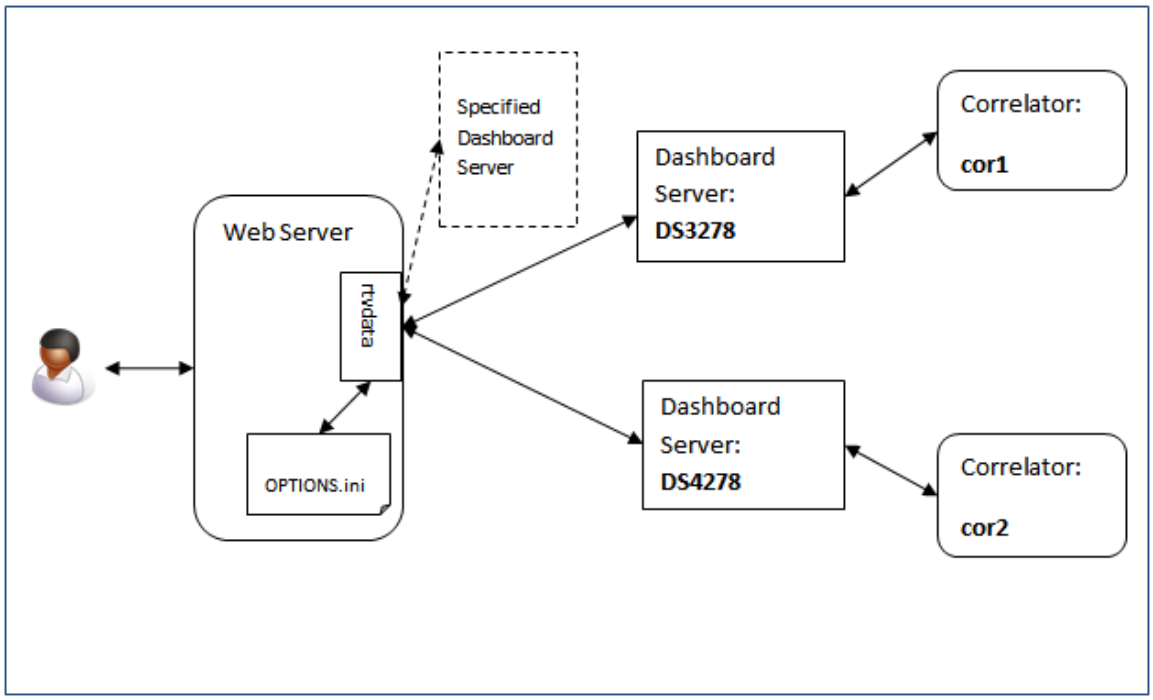
The logical data server names specified in the Builder's **Application Options** dialog are recorded in the file `OPTIONS.ini`, which is used by the Deployment Wizard to define deployment logical names. You can override these logical name definitions with the `--namedServer` *name* `:`*host* `:`*port* option to the Display Server executable. Below is an example. This is a sequence of command line options which should appear on a single line as part of the command to start the executable:

```
--namedServer Server1:ProductionHost_A:3278 --
namedServer Server2:ProductionHost_B:4278 --namedServer
Server3:ProductionHost_C:5278
```

Here `Server1`, `Server2` and `Server3` are the server logical names.

## Applet and WebStart deployments with multiple Data Servers

Applet and WebStart dashboards maintain connections with the Data Servers named in their attachments and commands.




In this diagram, the dotted line indicates the connection to the default Data Server, which is specified in the Startup and Server section of the Deployment Configuration Editor. The default must be running only if some attachments or commands don't specify a named Data Server.

The logical data server names specified in the Builder's **Application Options** dialog are recorded in the file `OPTIONS.ini`, which is used by the Deployment Wizard to define deployment logical names.

## Managing and stopping the Data Server and Display Server

The `dashboard_management` tool is used to stop a Data Server or Display Server and perform certain Data Server or Display Server management operations. The executable for this tool is located in the `bin` directory of the Apama installation. Running the tool in the Apama Command Prompt (see "Setting up the environment using the Apama Command Prompt" in *Deploying and Managing Apama Applications*) ensures that the environment variables are set correctly.

**Note:** The `component_management` tool can also be used for many of these tasks. See "Shutting down and managing components" in *Deploying and Managing Apama Applications*.

### Synopsis

To manage and stop the Data Server or Display Server, run the following command:

```
dashboard_management [ options ]
```

When you run this command with the `-h` option, the usage message for this command is shown.

### Description

You can use this tool to shut down, deep ping, or get the process ID of a Data Server or Display Server on a specified host and port. A successful deep ping verifies that the server is responding to requests. You can also use this tool to generate a dashboard deployment package, and to sign .jar files as part of deployment-package generation.

When you invoke this tool, you can specify the host and port of the server you want to manage. For the port, specify the port that was specified with the `-p` or `--port` option when the desired server was started. If the `-p` or `--port` option was not specified, you do not need to supply this option. It defaults to the default management port (28888).

### Options

The `dashboard_management` tool takes the following options:

| Option | Description |
|---|---|
| `-a` *alias* \| `--alias` *alias* | Use the alias in order to sign the `.jar` files to be included in the deployment package specified by the `-y` or `--deploy` option and the `-c` or `--config` option. Specify the keystore and password with the `-k` or `--keystoreFile` option and the `-o` or `--password` option. |
| `-c` *path* \| `--config` *path* | Generate a deployment package for the named configuration. Specify the file that defines the configuration with the `-y` or `--deploy` option. Specify the `.rtv` files to use with the `-r` or `--rtvPath` option. |
| `-D \| --displayServer` | Run against Display Server. |
| `-d \| --deepping` | Deep-ping the component. |
| `-e` *password* \| `--encryptString` *password* | Generate an encrypted version of the password. This is useful when you manually add an SQL data source by entering information directly into `OPTIONS.ini`. |
| `-f` *file* \| `--logfile` *file* | Full pathname of the file in which to record logging. |
| `-h \| --help` | Display usage information. |

| Option | Description |
|---|---|
| `-I \| --invalidateAll` | Invalidate all user authentications. |
| `-i` *`username`* `\| --invalidateUser` *`username`* | Invalidate a user authentication. |
| `-j` *`jarfile`* `\| --jar` *`jarfile`* | Name of a third-party jar file to sign. You can specify this option multiple times if you have multiple jar files to sign. |
| `-k` *`path`* `\| --keystoreFile` *`path`* | Use the keystore file designated by path in order to sign the `.jar` files to be included in the deployment package specified by the `-y` or `--deploy` option and the `-c` or `--config` option. Specify the alias and password with `-a` or `--alias` option and the `-o` or `--password` option. Ensure that the environment variable `JAVA_HOME` is set to a Java Development Kit (JDK). |
| `-l` *`path`* `\| --deployLocation` *`path`* | The deploy destination. |
| `-n` *`host`* `\| --hostname` *`host`* | Connect to component on host. If not specified, `localhost` is used. |
| `-o` *`password`* `\| --password` *`password`* | Use the specified password in order to sign the `.jar` files to be included in the deployment package specified by the `-y` or `--deploy` option and the `-c` or `--config` option. Specify the keystore and alias with the `-k` or `--keystoreFile` option and the `-a` or `--alias` option. |
| `-p` *`port`* `\| --port` *`port`* | Connect to component on port. Specify the port that was specified with the `-p` or `--port` option when the component was started. If the `-p` or `--port` option was not specified, you do not need to supply this option. It defaults to the default management port (28888). |
| `-r` *`path`* `\| --rtvPath` *`path`* | Generate a deployment package with the `.rtv` files in the directory designated by path. Specify the deployment configuration to use with the `-y` or `--deploy` option and the `-r` or `--rtvPath` option. |

| Option | Description |
|---|---|
| `-s reason \| --shutdown reason` | Shut down the component with the specified reason. |
| `-U path \| --update path` | Update the specified Release 2.4 `.rtv` file or files so that they are appropriate for use with this Apama release. *`path`* is the pathname of a file or directory. If path specifies a directory, all `.rtv` files in the directory are updated. |
| `-v \| --verbose` | Emit verbose output, including the startup settings (such as `dataPort` and `updateRate`) of the dashboard server to connect to. |
| `-V \| --version` | Display program name and version number and then exit. |
| `-W \| --waitFor` | Wait for component to be available. |
| `-y path \| --deploy path` | Generate a deployment package for a configuration defined in the dashboard configuration file designated by path. Specify the configuration name with the `-c` or `--config` option. Specify the `.rtv` files to use with the `-r` or `--rtvPath` option. |
| `--remoteDeploy options` | Deploy the dashboard war file to a remote host. *`options`* has the following form:<br>`war-file \| host:port \| admin:password \| type`<br>*`type`* can be one of the following: `WEBSTART`, `APPLET` or `DISPLAYSERVER`. |
| `--remoteUndeploy options` | Undeploy the dashboard from a remote host. *`options`* has the following form:<br>`dashboard \| host:port \| admin:password` |

# 24 Administering Dashboard Security


- Administering authentication ..... 678
- Authentication for local and WebSphere deployments ..... 678
- Administering authorization ..... 682
- Securing communications ..... 689
- Example: Implementing LoginModule ..... 689

Deployed dashboards are protected by the authorization and authentication facilities provided by Apama and your application server.

Apama's dashboard authentication facility prompts users for credentials before allowing any access to deployed dashboards. It gives you the ability to customize authentication by either configuring your application server to use the security realm and authentication service of your choice or by supplying any JAAS-supported authentication module as a plug-in to the Data Server or Display Server. See "Administering authentication" on page 678.

Apama's authorization facility includes access control that gives you the ability to control who can use a given dashboard. The facility also gives you the ability to control who can use dashboards to gain a given type of access to a given scenario, scenario instance, DataView definition, or DataView item. And it gives the ability to control who can send events from dashboards using the **Send Event** command. See "Administering authorization" on page 682.

In addition to authenticating and authorizing users, you need to consider how you will protect data sent to dashboards. This is discussed in "Securing communications" on page 689.

## Administering authentication

For dashboards deployed as simple Web pages, applets, or Web Start applications, authentication can be performed by your application server.

For dashboards deployed as local applications, or dashboards using the WebSphere application server, authentication is performed both at the dashboard and by the Data Server or Display Server. When a user starts the Dashboard Viewer, a login dialog appears, which prompts the user for a user name and password (as well as for the host and port of the Data Server or Display Server to connect to). The information entered is used to authenticate the user against the authentication service of your choice. Authenticated users are allowed to connect to the Server. See "Authentication for local and WebSphere deployments" on page 678 for more information.

Whenever a dashboard connects to or disconnects from a Data Server or Display Server, the server sends a special notification event to all correlators that are connected to it, provided that your project includes the Dashboard Support bundle. Your monitors or scenarios can make use of these events to implement further authentication-related administration. See "Managing Connect and Disconnect Notification" on page 667 in "Managing the Dashboard Data Server and Display Server" on page 653.

## Authentication for local and WebSphere deployments

Both the Dashboard Viewer and Data Server or Display Server provide authentication through the Java Authentication and Authorization Service (JAAS). JAAS provides a pluggable framework for user authentication and authorization.

The JRE provides authentication modules for use with common authentication services such as LDAP and Kerberos. It also supports the development of new authentication

modules for use with custom or proprietary authentication services. The Data Server, Display Server, and Viewer will work with any authentication module that supports JAAS. This openness allows you to integrate dashboards with your existing authentication service.

**Important:** Default authentication for local deployments uses a no-op implementation that supports the JAAS login module. All user name/password pairs are authenticated. You can customize authentication for local deployments by supplying your own implementation of the interface `javax.security.auth.LoginModule`.

## Dashboard Login Modules Provided by Apama

Apama provides the following JAAS login modules in the package `com.apama.dashboard.security`:

- `NoOpLoginModule`: Does no username or password validation. This is used by default for the Dashboard Builder, Viewer, Data Server, and Display Server.
- `UserFileLoginModule`: Loads user and role definitions from an XML file. See "Installing UserFileLoginModule" on page 681 for more information.
- `LdapLoginModule`: Authenticates against an LDAP service.

In addition, the Java Runtime Environment (JRE) includes several JAAS login modules:

- `JndiLoginModule`: The module prompts for a username and password and then verifies the password against the password stored in a directory service configured under JNDI.
- `KeyStoreLoginModule`: Provides a JAAS login module that prompts for a key store alias and populates the subject with the alias's principal and credentials.
- `Krb5LoginModule`: This login module authenticates users based on Kerberos protocols.

## Developing custom login modules

When developing your implementation of `LoginModule`, note that the Data Server and Display Server's built-in `CallbackHandler` currently recognizes only the `NameCallback` and `PasswordCallback`.

See "Example: Implementing LoginModule" on page 689 for a sample implementation of `LoginModule`, which you can also find under `samples\dashboard_studio\tutorial\src` in your Apama installation.

## Installing login modules

`NoOpLoginModule` is used by default for the Dashboard Builder, Viewer, Data Server, and Display Server. To change this, you must install the login module or modules that you want to use instead. To install login modules, do both of the following:

- Specify the login modules to use in the file `JAAS.ini` in the `lib` directory of your Apama installation.
- Create a `jar` file that includes your `LoginModule` implementation or implementations, and add the `jar` or its directory to `APAMA_DASHBOARD_CLASSPATH` (changes to this environment variable are picked up by dashboard processes only at process startup) or else add the `jar` or its directory to the list of **External Dependencies** in your project's **Dashboard Properties** (In Software AG Designer, right click on your project and select **Properties**, expand **Apama**, select **Dashboard Properties**, activate the **External Dependencies** tab, and click the **Add External** button). You can also use the `--dashboardExtraJars` command line argument to specify this `jar` file.

If your login module has dependencies on other `.jar` files, add these `.jar` files to the manifest of the login module `.jar` file.

Software AG Designer allows you to sign your `.jar` files when you create a deployment package. See "Preparing Dashboards for Deployment" on page 639.

**Note:** The login module you install can affect the Data Server or Display Server authorization behavior. If you install `UserFileLoginModule`, for example, the default Scenario authority will provide expanded access to users with the `apama_admin` role. For such users, it will grant view, edit, and delete access to all Scenario instances (in addition to granting such access to Scenario-instance owners). See "Providing a login module that supports a Scenario or Event Authority" on page 688 for more information.

If you are installing a login module provided by Apama (see "Dashboard Login Modules Provided by Apama" on page 679), you do not need to create a `jar` file as described above, as this class is provided with your Apama installation and is included in an existing `jar`.

`JAAS.ini` supports the standard JAAS configuration file format. Each entry in the file associates an application with a login module together with a specification of the module's parameter values. Here is a `JAAS.ini` that specifies the `UserFileLoginModule` for the Dashboard Viewer and Data Server applications:

```
/*
* Dashboard Builder security configuration.
*/
builder {
   com.apama.dashboard.security.NoOpLoginModule required
   debug=false;
};
/*
* Dashboard Viewer security configuration.
*/
viewer {
   debug=false;
   com.apama.dashboard.security.UserFileLoginModule required
       debug=false
       userFile="<INSTALL_PATH>\\etc\\dashboard-users.xml"
       refreshDelay="5000";
};
/*
* Dashboard DataServer security configuration.
```

```
*/
dataServer {
   debug=false;
   com.apama.dashboard.security.UserFileLoginModule required
       debug=false
       userFile="<INSTALL_PATH>\\etc\\dashboard-users.xml"
       refreshDelay="5000";
};
```

**Important:** Do not change the login module associated with the Dashboard Builder.

## Installing UserFileLoginModule

In a `JAAS.ini` file, you specify a module's parameter values with expressions of the form *formal-parameter=actual-parameter*. The login module `UserFileLoginModule` supports the following parameters:

- `debug`: `true` or `false`. Enable debug logging.
- `validateUser`: true or false. Set to false to disable password validation. This is for use in Web deployments where authentication is provide by your application server —see "Providing a login module that supports a Scenario or Event Authority" on page 688. In such a case, configure `userFile` to specify users for your application server. This results in the application server performing authentication and the Data Server handling authorization, in such a way that the application server and the Data Server obtain user-credentials information from the same file.
- `userFile`: Fully resolved name of the file with user and role definitions
- `refreshDelay`: Time in milliseconds to check for changes to the definition file (userFile). When it changes it is reloaded. This allows new users to be added.

**Note:** Installing `UserFileLoginModule`can affect the Data Server's authorization behavior. In particular, if you install `UserFileLoginModule`, the default Scenario authority will provide expanded access to users with the `apama_admin` role. For such users, it will grant view, edit, and delete access to all Scenario instances (in addition to granting such access to Scenario-instance owners).

## Installing LdapLoginModule

The `LdapLoginModule` uses the Java Naming and Directory Interface (JNDI) to access naming and directory services. Oracle's LDAP provider is supported, and hence the `InitialContext` must be set up based on Oracle's implementation. The environment settings must be specified in the `JAAS.ini` file. Here is an example:

```
viewer {
   com.apama.dashboard.security.LDAPLoginModule required
         ProviderURL="ldap://your.own.ldap.server:389"
         Authentication=simple
         Anonymous=false
         DN="uid=%,ou=City,ou=Region,ou=People,o=ACME Corporation"
         TLS=false;
};
```

## Administering authorization

Apama's dashboard authorization facility includes access control that gives you the ability to restrict who can use a given Web-based dashboard.

The example above configures the Dashboard Viewer to use `LdapLoginModule`.

Following are the supported environment settings:

- `ProviderURL` (required): Specifies the LDAP server and port, which are used to set the `java.naming.factory.initial` property.
- `Authentication` (required): Specifies the authentication mechanism to use. Specify `none`, `simple`, or `sasl_mech`. This value is used to set the `java.naming.security.authentication` property;
- `Anonymous` (optional; defaults to true): Specifies whether the `userPrincipal` and `userCredential` should be used when creating the `LdapContext`.
- `DN` (required): Specifies the user principal to be used when accessing the directory. This value is used (after patching with the user name) to set the `java.naming.security.principal` property. The user entered password is used in `java.naming.security.credentials`.

In the example above, `DN` is set to the following:

```
uid=%,ou=City,ou=Region,ou=People,o=ACME Corporation.
```

The `%` character is replaced by the login name entered by user.

- `TLS` (required): This specifies whether the LDAP server should start the Transport Security Layer extension. Supply `true` to specify that it should be started; supply `false` to specify that it should not be started.
- `Extra` (optional): Allows you to specify any extra parameters for setting the environment before creating the `LdapContext`. The function of these extra parameters is specific to your LDAP server, not the `LdapLoginModule`. Supply a semicolon-separated list of name/value pairs, where each pair has the form

*name=value*

Consider for example the following:

```
Extra=java.naming.referral=ignore;java.naming.security.protocol=ssl
```

This sets `java.naming.referral` to `ignore` and `java.naming.security.protocol` to `ssl`.

The facility also gives you the ability to control who can use dashboards for each of the following types of Scenario access:

- Viewing a given Scenario instance
- Editing a given Scenario instance
- Deleting a given Scenario instance

- Creating an instance of a given Scenario

In addition, you can control who can use dashboards for view access to DataView items. See "Default Scenario and DataView access control" on page 683 and "Customizing Scenario and DataView access control" on page 684.

You can also control who can send events from dashboards using the **Send Event** command. See "Send event authorization" on page 687

For Web deployments, some aspects of authorization (in particular, dashboard access control) are centered around the concepts of *users* and *roles*, which are introduced in "Users and roles" on page 683.

## Users and roles

A *user* is an individual in a company or organization, who proves their identity to the Application Server by entering a password known only to them.

A *role* defines a capability to perform an operation or access to some entity on the Application Server, and might typically be held by a number of users.

Each user has zero or more associated roles.

For Web-based deployments, you create users and roles, and associate roles with users by using your application server.

## Default Scenario and DataView access control

By default, only the owner of a Scenario instance or DataView item can access it, unless the owner is specified as "*", in which case all users can access it.

When a user attempts to access a Scenario instance by using a dashboard, view, edit, and delete access are authorized if and only if one of the following is true:

- The user name supplied during application-server or Data-Server login matches the Scenario-instance owner.
- The Scenario instance owner is specified as "*".

Similarly, when a user attempts to access a DataView item by using a dashboard, view access is authorized if and only if one of the following is true:

- The user name supplied during application-server or Data-Server login matches the DataView-item owner.
- The DataView-item owner is specified as "*".

By default, any user can create an instance of any Scenario.

Edit, create, and delete access do not apply to DataView items, but see Send Event Authorization.

## Customizing Scenario and DataView access control

You can customize Scenario and DataView access control by supplying a *Scenario Authority*, an implementation of the interface `com.apama.security.IScenarioAuthority`. This interface defines the methods `canView`, `canEdit`, `canDelete`, and `canCreate`, which must be implemented to return `true` or `false` for a given user and Scenario, Scenario instance, or DataView item. See "Providing a Scenario Authority" on page 684.

You might also need to supply a login module, an implementation of `javax.security.LoginModule`, in order to endow the instance of `javax.security.Subject` that represents the current end user with the characteristics that the Scenario or Event Authority requires. See "Providing a login module that supports a Scenario or Event Authority" on page 688.

## Providing a Scenario Authority

You can provide an alternative to the default Scenario Authority by doing one of the following:

- Develop and install a custom Scenario Authority. See "Developing a custom Scenario Authority" on page 684 and "Installing a Scenario Authority" on page 686.
- Install `com.apama.dashboard.security.NoOpScenarioAuthority`, which allows full access by any user. This is useful for testing. See "Installing a Scenario Authority" on page 686.

### *Developing a custom Scenario Authority*

In order to develop a custom Scenario Authority, you must implement each `IScenarioAuthority` method (see "Implementing IScenarioAuthority methods" on page 684), which typically requires the use of `UserCredentials` access methods (see "Using UserCredential accessor methods" on page 687). When you compile your implementation, your classpath must be set appropriately (see "Compiling your Event Authority" on page 687).

#### *Implementing IScenarioAuthority methods*

Below is a description of each `IScenarioAuthority` method that you must define, including the following:

- Signature of the method
- When the method is called by the Data Server
- What the method should return

When each method is called depends on whether *authorization caching* is on—see the `-r` or `--cacheUsers` option.

This interface defines the following methods:

```
public boolean canView (
```

```
  IUserCredentials credentials,
  IScenarioInstance instance
);
```

canView is called the first time (or, if authorization caching is off, every time) in a Data Server or Display Server session that the Server sends data from the specified Scenario instance or DataView item to an end user with the specified credentials. Your implementation should return `true` if the user with the specified credentials is authorized to view the specified instance or item; it should return `false` otherwise.

Note that if caching is off, `canView` is called very frequently, as specified by the update rate for the Data Server (see the description of the `-u` or `--updateRate` option). If your implementation renders calls to `canView` expensive, the performance of your dashboard will be significantly affected.

```
public boolean canEdit (
  IUserCredentials credentials,
  IScenarioInstance instance
);
```

canEdit is called the first time (or, if caching is off, every time) a dashboard attempts to edit a Scenario instance. Your implementation should return `true` if the user with the specified credentials is authorized to edit the specified Scenario instance; it should return `false` otherwise. Does not apply to DataView items, but see Send Event Authorization.

```
public boolean canDelete (
  IUserCredentials credentials,
  IScenarioInstance instance
);
```

canDelete is called the first time (or, if caching is off, every time) a dashboard attempts to delete a Scenario instance. Your implementation should return `true` if the user with the specified credentials is authorized to delete the specified Scenario instance; it should return `false` otherwise. Does not apply to DataView items, but see Send Event Authorization.

```
public boolean canCreate (
  IUserCredentials credentials,
  IScenarioDefinition scenario
);
```

canCreate is called the first time (or, if caching is off, every time) a dashboard attempts to create a Scenario. Your implementation should return `true` if the user with the specified credentials is authorized to create an instance of the specified Scenario; it should return `false` otherwise. Does not apply to DataViews, but see Send Event Authorization.

#### Using UserCredential Accessor Methods

Your implementation of `IScenarioAuthority` will typically use the following public assessor methods of `com.apama.dashboard.security.IUserCredentials`:

```
public String getUsername()
public String getPassword()
public Subject getSubject()
```

Your implementation may also use the following methods of com.apama.services.scenario.IScenarioInstance:

```
public String getOwner()
```

```
public Object getValue(String parameterName)
```

#### Compiling Your Scenario Authority

When you compile your implementation of `IScenarioAuthority`, be sure to put the following `jar` files on your classpath:

- `ap-dashboard-client.jar`
- `ap-client.jar`

These `jar` files are in the `lib` directory of your Apama installation.

#### *Installing a Scenario Authority*

To install your authorization customization, do both of the following:

- Replace "`com.apama.dashboard.security.DefaultScenarioAuthority`" with the fully qualified name of your class in the file `EXTENSIONS.ini`, which is in the `lib` directory of your Apama installation.
- Create a `jar` file that contains your `IScenarioAuthority` implementation, and add the `jar` or its directory to `APAMA_DASHBOARD_CLASSPATH` (changes to this environment variable are picked up by dashboard processes only at process startup) or else add the `jar` or its directory to the list of **External Dependencies** in your project's **Dashboard Properties** (In Software AG Designer, right click on your project and select **Properties**, expand **Apama**, select **Dashboard Properties**, activate the **External Dependencies** tab, and click the **Add External** button). You can also use the `--dashboardExtraJars` *`jarFiles`* command line argument to specify this `jar` file.

If your scenario authority has dependencies on other `.jar` files, add these `.jar` files to the manifest of the scenario authority `.jar` file.

Software AG Designer allows you to sign your `.jar` files when you create a deployment package. See "Preparing Dashboards for Deployment" on page 639.

If you are installing `NoOpScenarioAuthority`, you do not need to create a `jar` file as described above, as this class is provided with your Apama installation and is included in an existing `jar`.

The `EXTENSIONS.ini` specifies the scenario authority to use. This file identifies all the user supplied extension classes (including functions and commands). Here is a sample `EXTENSIONS.ini`:

```
function com.apama.dashboard.sample.SampleFunctionLibrary
command com.apama.dashboard.sample.SampleCommandLibrary
scenarioAuthority com.apama.dashboard.sample.SampleScenarioAuthority
```

This file installs a function library, a command library, and a Scenario authority.

If multiple authorities are specified, a user must be authorized by each.

## Sample Custom Scenario Authority

You can find a sample implementation of `IScenarioAuthority` under `samples\dashboard_studio\tutorial\src`:

## Send event authorization

By default, any user is authorized to send any event. However you can create a custom *event authority* that determines whether a given user is authorized to send a given event. An event authority is a Java class that implements the `canSend` method of the interface `com.apama.dashboard.security.IEventAuthority`:

```
boolean canSend (IUserCredentials credentials, Event event);
```

If `canSend()` returns `true` the user is allowed to send the event. If it returns `false` the user is not allowed to send the event and the attempt to send the event is treated as a command failure. Dashboard object property settings determine if this error is displayed to the user.

### *Using UserCredential accessor methods*

Your implementation of `IEventAuthority` will typically use the following public assessor methods of `com.apama.dashboard.security.UserCredentials`:

```
public String getUsername()
public String getPassword()
public Subject getSubject()
```

### *Compiling your Event Authority*

When you compile your implementation of `IEventAuthority`, be sure to put the following `jar` files on your classpath:

- `ap-dashboard-client.jar`
- `ap-client.jar`

These `jar` files are in the `lib` directory of your Apama installation.

### *Installing your Event Authority*

To install your authorization customization, do both of the following:

- Replace `com.apama.dashboard.security.NoOpEventAuthority` with the fully qualified name of your class in the file `EXTENSIONS.ini`, which is in the `lib` directory of your Apama installation.
- create a `jar` file that contains your `IEventAuthority` implementation, and place it in the directory `%APAMA_WORK%\dashboards_deploy\lib`. If you have other custom classes (for example, a custom login module—see "Developing custom login modules" on page 679), you can include them in the same `.jar` file or in a different `.jar` file.

If your event authority has dependencies on other `.jar` files, add these `.jar` files to the manifest of the event authority `.jar` file.

Software AG Designer allows you to sign your `.jar` files when you create a deployment package — see *Preparing Dashboards for Deployment* in *Building and Using Dashboards* .

Two event authorities are provided with your installation:

- `com.apama.dashboard.security.NoOpEventAuthority`: Permits all users to send any event.
- `com.apama.dashboard.security.DenyAllEventAuthority`: Denies all users rights to send any event.

`NoOpEventAuthority` is the default event authority. Use a custom event authority when deploying your dashboards.

If you are installing `DenyAllEventAuthority`, you do not need to create a `jar` file as described above, as this class is provided with your Apama installation and is included in an existing `jar`.

Here is a portion of `EXTENSIONS.ini` as shipped:

```
# List of event authorities. An event authority is called to determine
# if a user has rights to send an event to a correlator. Each must implement
# the interface:
##     com.apama.dashboard.security.IEventAuthority
## Multiple authorities can be specified. They will be called in the order
# listed.
## Format:
##    eventAuthority <classname>
## NoOpEventAuthority - Allows all users to send events
eventAuthority com.apama.dashboard.security.NoOpEventAuthority
# DenyAllEventAuthority - Allows no users to send events
#eventAuthority com.apama.dashboard.security.DenyAllEventAuthority
# eventAuthority <your_class_name_here>
```

## Providing a login module that supports a Scenario or Event Authority

When you implement a Scenario or Event Authority, the methods that you implement have a `UserCredentials` argument. Typical implementations retrieve an instance of `javax.security.auth.Subject` from the `UserCredentials` object, and use the `Subject`'s characteristics to determine whether to return `true` or `false` (that is, whether to grant or deny access).

The characteristics of a particular Subject (for example its associated roles, as returned by `Subject.getPrinciples`) are established by a JAAS login module that is called by the Data Server or Display Server. It is this module's responsibility to establish those characteristics on which the Scenario or Event Authority will rely.

For local deployments, this login module is responsible for authenticating the current end user (see "Authentication for local and WebSphere deployments" on page 678) as well as for setting the characteristics of the `Subject`. For Web deployments, this login module is responsible only for setting the characteristics of the `Subject` (since authentication is performed by application server.

For both Web and local deployments, the default Data Server and Display Server login module is `NoOpLoginModule`, which does *not* set any characteristics of the `Subject`. With this module, the `Subject` passed into `IScenarioAuthority` methods has no associated roles.

Typical implementations of `LoginModule` store the `Subject` passed into `LoginModule.initialize` as local state, and set the `Subject`'s characteristics in `LoginModule.commit`.

Note that `UserFileLoginModule` supports Scenario Authorities by setting `Subject` roles at the time of authentication. To use `UserFileLoginModule` in order to support Scenario Authorities for Web-based deployments (where authentication is performed by the application server), set `validateUser` to false when you install `UserFileLoginModule`—see "Installing UserFileLoginModule" on page 681.

For Web-based deployments, the Data Server and Display Server receive only user names (and not passwords) from the application server. This means that you cannot use a JAAS login module that requires both user names and passwords in order to authenticate users and retrieve their roles. To perform role based authorization for Web-based deployments, use a JAAS login module that can retrieve the roles for a user by using only the user name.

## Securing communications

For local application deployments, where dashboards communicate directly with the Data Server, your options for securing dashboard communications include:

- Enabling secure sockets (SSL) in the Data Server
- Utilizing a secure channel (SSH) for all dashboard communication
- Utilizing a virtual public network (VPN) for all dashboard access

For applet or Web Start deployments, where dashboards communicate through an application server, your options include the following:

- Enabling HTTPS in the application server
- Utilizing a secure channel (SSH) for all dashboard communication
- Utilizing a virtual public network (VPN) for all dashboard access.

As with all encryption technology, there is a cost in performing the encryption and decryption of data. For applications with a very high frequency of data changes, you should account for this cost when you determine how frequently a dashboard can be updated (see the description of the `-u` or `--update` option).

## Example: Implementing LoginModule

Below is a sample implementation of `LoginModule`, which you can find under `samples\dashboard_studio\tutorial\src`. See "Authentication for local and WebSphere deployments" on page 678.

```
package com.apama.dashboard.sample;
import java.security.Principal;
import java.util.HashSet;
import java.util.Map;
import java.util.Set;
```

```
import javax.security.auth.Subject;
import javax.security.auth.callback.Callback;
import javax.security.auth.callback.CallbackHandler;
import javax.security.auth.callback.NameCallback;
import javax.security.auth.callback.PasswordCallback;
import javax.security.auth.callback.UnsupportedCallbackException;
import javax.security.auth.login.FailedLoginException;
import javax.security.auth.login.LoginException;
import javax.security.auth.spi.LoginModule;
/**
 * SampleLoginModule is an example of a custom JAAS login module for
 * Dashboard Builder. Custom JAAS login modules allow you to extend Dashboard
 * Builder to use the authentication mechanism of your choosing.
 * <p>
 * SampleLoginModule authenticates all users, regardless of username
 * and password and adds the Principal "apama_customer" to each.
 *
 * $Copyright(c) 2013 Software AG, Darmstadt, Germany and/or its licensors$
 *
 * @version      $Id: SampleLoginModule.java 84623 2008-06-25 22:41:10Z cr $
 */
public class SampleLoginModule implements LoginModule {
  // Option strings
  private final static String OPT_DEBUG = "debug";
  // Initial state
  private Subject subject;
  private CallbackHandler callbackHandler;
  // True if debug logging turned on
  private boolean debug = false;
  // Authentication status
  private boolean succeeded = false;
  private boolean commitSucceeded = false;
  // Username and password
  private String username;
  private char[] password;
  /**
   * Initialize this LoginModule.
   *
   * @param subject Subject to be authenticated
   * @param callbackHandler CallbackHandler for communicating with the user to
   * obtain username and password
   * @param sharedState Shared LoginModule state
   * @param options Options specified in the login Configuration for this
   * LoginModule.
   */
  public void initialize(
    Subject subject, CallbackHandler callbackHandler, Map sharedState,
     Map options) {
    this.subject = subject;
    this.callbackHandler = callbackHandler;
    // Process options
    debug = "true".equalsIgnoreCase((String) options.get(OPT_DEBUG));

    // Add additional options and initialization here

    // Must have a callback handler
    if (callbackHandler == null) {
      throw new IllegalArgumentException (
        "Error. Callback handler must be specified.");
    }
  }
  /**
   * Authenticate the user by calling back for username and password.
```

```
  *
  * @return true in all cases
  * @exception FailedLoginException if the authentication fails
  * @exception LoginException if unable to perform the authentication
  */
 public boolean login() throws LoginException {
   // Callback to get username and password
   Callback[] callbacks = new Callback[2];
   callbacks[0] = new NameCallback("username: ");
   callbacks[1] = new PasswordCallback("password: ", false);
   try {
     // Perform the callbacks
     callbackHandler.handle(callbacks);

     // Get the user supplied name and password
     username = ((NameCallback) callbacks[0]).getName();
     password = ((PasswordCallback) callbacks[1]).getPassword();
     if (password == null) {
       // Treat a NULL password as an empty password
       password = new char[0];
     }

     // Clear password
     ((PasswordCallback) callbacks[1]).clearPassword();
   } catch (java.io.IOException e) {
     throw new LoginException("UserFileLoginModule. Error performing
           callbacks. " +
       e.toString());
   } catch (UnsupportedCallbackException e) {
     throw new LoginException("UserFileLoginModule. Error performing
           callbacks " +
                           e.getCallback().toString() + ".");
   }
   // verify the username/password
   if (validateUser()) {
     if (debug) {
       System.err.println("UserFileLoginModule.
           User authenticated: " + username);
     }
     succeeded = true;
     return true;
   } else {
     if (debug) {
       System.err.println("UserFileLoginModule. User authentication failed: " +
         username);
     }
     succeeded = false;
     clearState();
     throw new FailedLoginException("UserFileLoginModule. Login failed.");
   }
 }
 /**
  * Called if the LoginContext's overall authentication succeeded
  * (the relevant REQUIRED, REQUISITE, SUFFICIENT and OPTIONAL
  * LoginModules succeeded).
  * <p>
  * Add the user's principals (roles) to the Subject
  *
  * @exception LoginException Commit failed
  * @return true if commit attempts succeeded; false otherwise.
  */
 public boolean commit() throws LoginException {
   if (succeeded == false) {
```

```
      return false;
    } else {

      // Get the users roles from the user database and add each as a
      // SimplePrincipal
      subject.getPrincipals().addAll(getUserPrincipals());
      clearState();
      commitSucceeded = true;
      return true;
    }
  }
  /**
   * Called if the LoginContext's overall authentication
   * failed. (the relevant REQUIRED, REQUISITE, SUFFICIENT and OPTIONAL
   * LoginModules did not succeed).
   * <p>
   * Cleans state information.
   *
   * @exception LoginException Abort failed
   * @return true if abort successfule; false otherwise
   */
  public boolean abort() throws LoginException {
    if (succeeded == false) {
      return false;
    } else if (succeeded == true && commitSucceeded == false) {
      // login succeeded but overall authentication failed
      succeeded = false;
      clearState();
    } else {
      // overall authentication succeeded and commit succeeded,
      // but another LoginModule's commit failed
      logout();
    }
    return true;
  }
  /**
   * Logout the user.
   *
   * @return true in all cases
   * @exception LoginException Logout failed
   */
  public boolean logout() throws LoginException {
    succeeded = false;
    succeeded = commitSucceeded;
    clearState();
    return true;
  }
  /**
   * Clear out temporary state used in a single login attempt.
   */
  private void clearState () {
    username = null;
    if (password != null) {
      for (int i = 0; i < password.length; i++)
        password[i] = ' ';
      password = null;
    }
  }
  /**
   * Validate current username/password pair.
   *
   * @return true if validated.
   */
```

```
  private boolean validateUser () {

    //
    // Add user validation here.
    //
    System.out.println("Validate username: " + username + " password: " +
      new String(password));
    return true;
  }

  /**
   * Get the principals (roles) for the current username.
   *
   * @return Set of Principals
   */
  private Set<Principal> getUserPrincipals () {
    HashSet<Principal>  set = new HashSet<Principal>();
    //
    // Add user principals here.
    //

    System.out.println("Add principal username: " + username +
      " principal: apama_customer");
    set.add (new SamplePrincipal("apama_customer"));
    return set;
  }
}package com.apama.dashboard.sample;
import java.security.Principal;
/**
 * SamplePrincipal is an example of a Java security principal (role) for
 * use with JAAS authentication.
 * <p>
 * SamplePrincipal provides a simple string based principal.
 *
 * $Copyright(c) 2013 Software AG, Darmstadt, Germany and/or its licensors$
 *
 * @version      $Id: SamplePrincipal.java 84623 2008-06-25 22:41:10Z cr $
 */
public class SamplePrincipal implements Principal {
  // Principal name
  private String name;
  /**
   * Constructor.
   *
   * @param name Principal name
   */
  public SamplePrincipal(String name) {
    this.name = name;
  }
  /**
   * Get the name of the principal.
   */
  public String getName() {
    return name;
  }
}
```

# V Using the Dashboard Viewer


- Concepts Underlying Dashboards ..... 697
- Using the Dashboard Viewer ..... 701
- Startup Options for the Dashboard Viewer ..... 709
- Timezone ID Values ..... 715

This section describes how to use the Apama Dashboard Viewer, which is the runtime viewer for local deployments of Apama dashboards. It provides the ability to view and interact with dashboards that are receiving live data from an Apama Dashboard Server. Dashboard Servers serve dashboards for applications running in Apama correlators.

This section assumes that you have already installed Apama.

The information in this section is organized as follows:

- "Concepts Underlying Dashboards" on page 697 introduces the concepts underlying Dashboards and their runtime usage.
- "Using the Dashboard Viewer" on page 701 describes how to use the various objects included in a Dashboard Viewer.
- "Startup Options for the Dashboard Viewer" on page 709 provides advanced information on starting Dashboard Viewers.
- "Timezone ID Values" on page 715 lists the timezone ID values used when manually starting the Dashboard Viewer as described in "Startup Options for the Dashboard Viewer" on page 709.

# 25 Concepts Underlying Dashboards


- About Dashboards ..... 698
- Starting the Dashboard Viewer ..... 699
- Scenario instance and DataView item ownership ..... 700

This guide assumes that you have already installed the Dashboard Viewer. This chapter introduces the concepts underlying dashboards and their runtime use. "Using the Dashboard Viewer" on page 701 describes how to use the various visualization objects that are included in dashboards. "Startup Options for the Dashboard Viewer" on page 709 provides advanced information on starting the Dashboard Viewer.

## About Dashboards

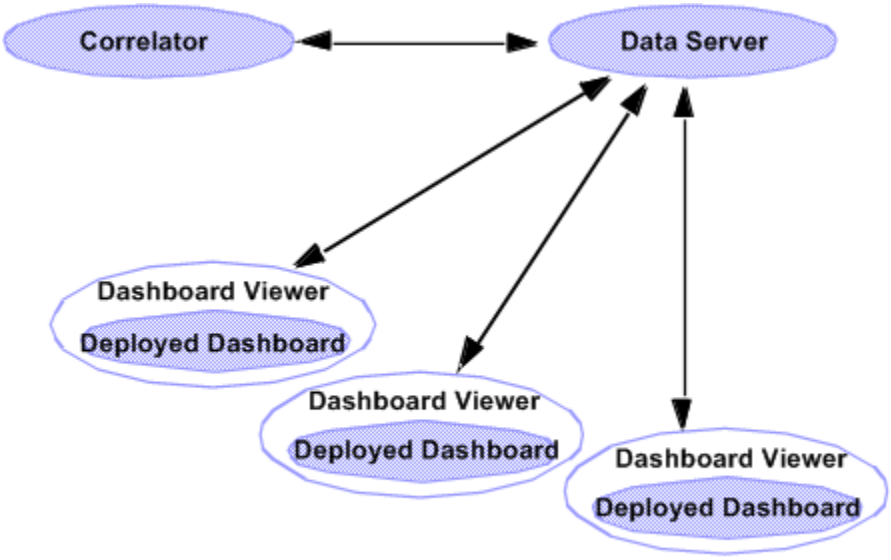
Dashboards provide the ability to view and interact with applications running in a correlator. Dashboards contain charts and other objects that allow you to visualize the status, performance, and attributes of Apama applications including any scenarios and DataViews. Dashboards can also contain control objects for creating, editing, and deleting scenario instances, as well as for sending events to a correlator.

Dashboard displays are stored in `.rtv` files. A dashboard project includes `.rtv`, image, and audio files. A dashboard project is deployed in a single directory with one or more subdirectories containing the files of the project. To use a dashboard, the Dashboard Viewer must have access to all the files in the associated dashboard-project directory.

Deployed dashboards connect to a dashboard Data Server, which in turn connects to one or more correlators. The dashboard Data Server is the middle-tier between users and the correlator. It provides for both scalability and security. As the scenarios or DataViews in a correlator run, and their variables or fields change, update events are sent to dashboard Data Servers, which in turn send the data to all connected dashboards. When a dashboard receives an update event, it updates its display in real time to show the behavior of the application. User interactions with the dashboard, such as creating an instance of a scenario, result in control events being sent via the Data Server to the correlator.

The following diagram illustrates the runtime deployment of dashboards:

**Runtime deployment of dashboards**




With the Dashboard Viewer, dashboards communicate with the Dashboard Data Server, which communicates with the correlator.

## Starting the Dashboard Viewer

The simplest way to start the Dashboard Viewer is from the Windows **Start** menu. Select **All Programs > Software AG > Tools > Apama *n.n* > Apama Dashboard Viewer *n.n***.

When you start the Viewer, the Data Server login prompt appears.

By default, you can log in with any user name and password, but your user name must match the owner of any scenario instances or DataView items that you want to view. Your dashboard administrator might have implemented a non-default authentication and authorization scheme.

The recommended deployment for the Dashboard Viewer is through a dashboard Data Server. The **Connect directly to correlator** checkbox allows you to connect directly to a correlator without the use of a Data Server. This is not recommended for live deployments, as it is not secure and not as scalable as connections via the Data Server.

If all attachments and commands use named Data Servers, you can check the **Only using named data server connections** check box and omit specification of a default server.

The Dashboard Viewer can also be started by running the `dashboard_viewer` executable, located in the Apama `bin` directory. This method of starting the viewer is useful when passing start-up options on the command line. The start-up options supported by Dashboard Viewer are detailed in "Startup Options for the Dashboard Viewer" on page 709.

## Scenario instance and DataView item ownership

Scenario instances and DataView items in a correlator include an attribute identifying the owner of the instance. When a scenario instance is created through Dashboard Builder, it provides the current user ID as the owner of the instance.

When viewing scenario instances or DataView items in Dashboard Builder, you are by default only allowed to see and operate on those instances or items that you own; that is, by default the current user ID must match the owner attribute of the instance or item. Your dashboard administrator might have implemented a non-default authorization scheme.

# 26 Using the Dashboard Viewer


- Opening and viewing dashboards ..... 702
- The Dashboard Viewer menu bar ..... 703
- Resizing the Dashboard Viewer ..... 705
- Working with Dashboard Objects ..... 705

"Concepts Underlying Dashboards" on page 697 introduces the important concepts underlying the Dashboard Viewer, and describes how to start the Viewer. This chapter illustrates how to use the Dashboard Viewer.

By default, no dashboard is displayed. This chapter describes how to open and work with dashboards.

## Opening and viewing dashboards

The Dashboard Viewer main window can open and display one dashboard at a time.

To open a dashboard, select **File > Open** from the Viewer menu and select the `.rtv` file you want to open.

### Running the Statistical Arbitrage demo

The examples in this chapter use the Statistical Arbitrage demo dashboard. Although it is not necessary to run the Statistical Arbitrage demo, you may find it useful to do so.

You can run the Statistical Arbitrage demo as follows:

1. Start Software AG Designer.
2. From the **Help** menu, choose **Welcome**.
3. Under the **Apama** heading, click **Demos**.
4. Select the Statistical Arbitrage demo.
5. Click the **Open** button.
6. Click the play button in the **Launch Control Panel**.

Once you have launched the Statistical Arbitrage demo, a Statistical Arbitrage dashboard appears automatically (the dashboard to display was passed by Software AG Designer when the demo was launched).

### Displaying additional dashboards

A dashboard project can consist of more than one dashboard. In many cases, each dashboard is displayed one at a time, in the Dashboard Viewer main window. In other cases, separate windows are created to display additional dashboards.

Displaying dashboards in separate windows is common for dashboards that are used to create or edit scenario instances. For example, to see the separate dashboard used to create scenario instances in the Statistical Arbitrage demonstration, click the **Create** button in the Statistical Arbitrage dashboard. This displays a separate dashboard in a new window.

Any dashboard can be designed to display other dashboards in separate windows. The dashboards may even be nested; for example, the **Create** window in the Statistical Arbitrage sample could itself have been designed to display additional windows.

Window usage is specified when the dashboard project is created in the Dashboard Builder.

### Creating scenario instances visualized by the Statistical Arbitrage main dashboard

The **Create** dashboard in the Statistical Arbitrage dashboard project is subordinate to the main dashboard; it is intended to be accessed only from the main dashboard. Although it is possible to open the **Create** dashboard directly in Dashboard Viewer main window, you should not do so. Subordinate dashboards are typically dependent on context created by the parent dashboard and should only be accessed as intended by the creator of the dashboard project.

**To create scenario instances that are visualized by the Statistical Arbitrage main dashboard**

1. Accept the defaults and click the **Create** button in the **Create** dashboard.
2. Click the Create button in the main dashboard again.
3. Enter MSFT in the **Instrument1** field and **ORCL** in the **instrument2** field.
4. Click on a row of the Summary of All Strategies table in the main dashboard.

The data used for charting is stored in the Dashboard Viewer. As the Viewer runs, it accumulates historical data for display in charts. If you exit and restart the Dashboard Viewer, previously displayed historical data is not available.

## The Dashboard Viewer menu bar

There are four menus on the menu bar. Each has a number of nested menu options.

| Menu > Command | Description |
|---|---|
| **File** | All operations related to opening, printing, and closing dashboards |
| **File > Open** | Open a dashboard. |
| **File > Print** | Print the contents of a dashboard. |
| **File > Exit** | Exit the Dashboard Viewer. |
| **View** | All operations that manipulate the dashboard view. |
| **View > Zoom In** | Zoom in on a location in the dashboard. This switches the pointer to zoom mode, as indicated by the pointer changing to a crosshair +. In this mode, you can click |

| Menu > Command | Description |
|---|---|
| | on an area of the dashboard to zoom in on it and display it in greater detail. Right click to exit zoom mode. |
| **View > Zoom Out** | Zoom out on a location in the dashboard. This switches the pointer to zoom mode, as indicated by the pointer changing to a crosshair +. In this mode you can click on an area of the dashboard to zoom out on it and display it in less detail. Right click to exit zoom mode. |
| **View > Zoom Rect** | Zoom in on an area of the dashboard. This switches the pointer to zoom mode, as indicated by the pointer changing to a crosshair +. In this mode you can click and drag to select an area of the dashboard to zoom in on. Right click to exit zoom mode. |
| **View > Pan** | Pan the dashboard to show areas not currently displayed. This switches the pointer to pan mode, as indicated by the pointer changing to the pan pointer  . In this mode you can click and drag the dashboard to reveal areas not displayed. Right click to exit pan mode. It is not possible to pan if 100% of the dashboard view is visible. |
| **View > 100%** | Make the entire dashboard visible. |
| **Tools** | Change preferences |
| **Tools> Pause Display** | Pause the automatic updating of the dashboard. When not paused the dashboard automatically updates as data changes; when paused, updating does not occur. When paused, clicking on the dashboard will cause it to update. |
| **Help > Help Contents** | Displays the Apama documentation for your installation. |
| **Help > Command Line Options** | Displays a list of the Viewer options that you can supply at the command line. |
| **Help > About** | Displays information about this version of the Dashboard Viewer. |

## Resizing the Dashboard Viewer

When a dashboard is created in the Dashboard Builder, the Builder specifies a width and height for the dashboard. You can resize Dashboard Viewer windows, but the aspect ratio of width to height cannot be altered. If you resize a window to a different aspect ratio, the window size will automatically be adjusted in order to maintain the aspect ratio specified in the Builder.

When you resize a Dashboard Viewer window, the objects within it are scaled in order to maintain their size relative to the size of the window. Scaling allows dashboards to be enlarged in order to allow greater detail to be displayed, or reduced so that the dashboard occupies a smaller area of the screen.

When a dashboard Window is reduced in size, objects such as charts will scale all their visual elements in order to maintain proper appearance at the new size. Other objects, such as tables and input controls, adjust their width and height but may not scale all their visual elements, such as fonts in table column headers.

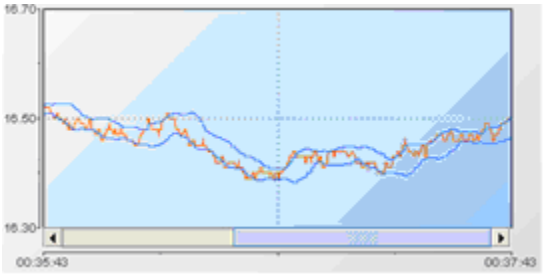
## Working with Dashboard Objects

Many of the objects displayed in a dashboard are familiar user interface controls. Their operations will not be covered in this guide. The topics below briefly introduce some of the objects that may not be familiar and that are used for the visualization of complex data.

### Trend charts

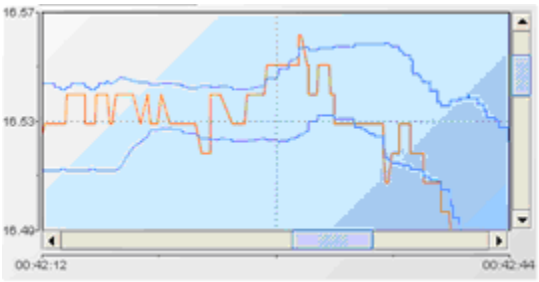
Trend charts provide the ability to view the performance of one or more scenario instances or DataViews items over time.

#### Trend chart




If enabled in the Dashboard Builder, trend charts support the ability to zoom in on an area of the chart. To zoom in on an area of a trend chart, click on the chart and drag the pointer to draw a box around the area to be zoomed.

#### Zooming in a trend chart area



To zoom out of a chart, hold down the **Shift** key while clicking on the chart.

If enabled in the Dashboard Builder, trend charts support scrolling to view historical values outside the scope of what fits in the trend chart window. Use the horizontal scroll bar to view older values.

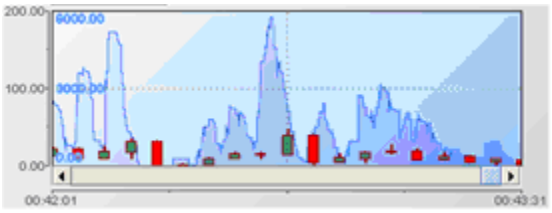
The number of values stored in historical data is limited. The limit is defined in the `OPTIONS.ini` file and can be overridden at startup by specifying options as detailed in "Startup Options for the Dashboard Viewer" on page 709.

When the maximum number of values is reached, the Dashboard Viewer begins to remove the oldest values in order to make room for the newest values. When the maximum number of values is reached, you see the oldest values being removed from the end of the trend chart.

## Stock charts

Stock charts provide the ability to view open, high, low, and close values, at a specified time interval, for a variable of a scenario instance or field of a DataView item. Stock charts support the same zooming, scrolling, and maximum-number-of-values behavior as trend charts.

#### Stock chart



## Tables

Tables provide the ability to view variable or field values for multiple scenario instances or DataView items. They are often used for summary displays of scenario instances or DataView items.

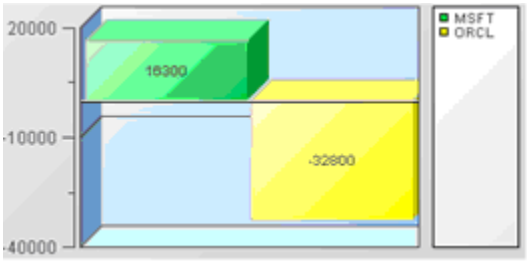
### Table

| Symbol1 | Symbol2 | Current Position1 | Current Position2 | Trades Executed | Status Message |
|---|---|---|---|---|---|
| CBS | VIA | -150000 | 249500 | 42954 | Limits Exceeded. Scenario... |
| MSFT | ORCL | 400 | -800 | 32880 | Submitting Orders |

Dashboard tables support many common table operations, such as sorting, column resizing, and column ordering. If enabled in the Builder, a table may also support drilldown to display detailed information about a scenario instance or DataView item. To drilldown on an instance or item that is displayed in a table, click on it.

## Pie and Bar charts

Pie and Bar charts are typically used to display summary information about one or more scenario instances.




If enabled in the Builder, a pie or bar chart may also support drilldown to display detailed information about a scenario instance. To drill down on an instance or item that is displayed in a pie or bar chart, click on it.

# 27 Startup Options for the Dashboard Viewer

The Dashboard Viewer supports options that can be specified on the start-up command line to override the default values used by the viewer. The executable for the Dashboard Viewer is `dashboard_viewer`, which can be found in the `bin` directory of the Apama installation.

### Synopsis

To pass start-up options to the Dashboard Viewer, run the following command:

```
dashboard_viewer [ options ] [ rtv-file-path ]
```

When you run this command with the `-h` option, the usage message for this command is shown.

### Options

The `dashboard_viewer` executable takes the following options:

| Option | Description |
|---|---|
| `-C file` \| `--panelConfig file` | Specifies the panels-initialization file to use for displaying this dashboard. `file` is the full pathname of the `.ini` file. |
| `-c logical-name:host:port:bool` \| `--correlator logical-name:host:port:bool` | Sets the correlator host and port for a specified logical correlator name. `bool` is one of `true` and `false`, and specifies whether to use the raw channel for communication. This overrides the host, port, and raw-channel setting specified by the Dashboard Builder for the given correlator logical name; see "Changing correlator definitions for deployment" on page 641. You can specify connection details for multiple correlators by using the option multiple times in a single command. Here is an example:<br>`-c default:localhost:15903:false -c work1:somehost:19999:true`<br>These options set the host and port for the logical names `default` and `work1`. |
| `-D` \| `--nodirect` | Suppress the check box in the login dialog that allows a direct connection to the correlator. |

| Option | Description |
|---|---|
| `-d \| --direct` | Connect directly to the correlator. |
| `-E bool \| --purgeOnEdit bool` | Specifies whether to purge all trend data when a scenario instance is edited. *bool* is one of `true` and `false`. If this option is not specified, all trend data is purged when an instance is edited. In most cases, this is the desired mode of operation. |
| `-F arg \| --filterInstance arg` | Exclude instances which are not owned by the user. This option applies to all dashboard processes. Default is `false` for builder and `true` for the other dashboard processes.<br><br>Exception: when the Dashboard Viewer is connecting to a Dashboard Server, the default is `true` and cannot be overridden. |
| `-f file \| --logfile file` | Full pathname of the file in which to record logging. If this option is not specified, the options in the log4j properties file will be used. |
| `-G file \| --trendConfigFile file` | Trend configuration file for controlling trend-data caching. |
| `-h \| --help` | Emit usage information and then exit. |
| `-I \| --xmlRedirect` | Redirect XML sources through Data Server. |
| `-J file \| --jaasFile file` | Full pathname of the JAAS initialization file to be used by the Data Server. If not specified, the Data Server uses the file `JAAS.ini` in the `lib` directory of your Apama installation. |
| `-j \| --singleClick` | Single click for drill down and actions. |
| `-k \| --doubleClick` | Double click for drill down and actions. |
| `-L file \| --xmlSource file` | XML data source file. If *file* contains static data, append `:0` to the file name. This signals Apama to read the file only once. |
| `-m mode \| --connectMode mode` | Correlator-connect mode. *mode* is one of `always` and `asNeeded`. If `always` is specified, all correlators are connected to at startup. If `asNeeded` is specified, |

| Option | Description |
|---|---|
| | correlators are connected as needed. If this option is not specified, the Data Server connects to correlators as needed<br><br>When `--connectMode always` is specified, trend data starts collecting upon correlator connection. When `--connectMode asNeeded` is in effect, trend data starts collecting after you select an attachment to the trend table in the Dashboard Viewer. |
| `-N` *`name`* \| `--name` *`name`* | Set the component name for identification in the correlator. The default name is `Dashboard Viewer:` *`username`*. |
| `-B` *`logical-name:host:port`* \| `--namedServer` *`logical-name:host:port`* | Sets the host and port for a specified logical Data Server name. This overrides the host and port specified by the Dashboard Builder for the given server logical name. This option can occur multiple times in a single command. |
| `-n` \| `--noSplash` | Do not display splash screen in startup. |
| `-e` \| `--noMenus` | Do not display menu bar. |
| `-O` *`file`* \| `--optionsFile` *`file`* | Use the specified `OPTIONS.ini` file at startup. |
| `-P` *`n`* \| `--maxPrecision` *`n`* | Maximum number of decimal places to use in numerical values displayed by dashboards. Specify values between 0 and 10, or -1 to disable truncation of decimal places. A typical value for *`n`* is 2 or 4, which eliminates long floating point values (for example, 2.2584435234). Truncation is disabled by default. |
| `-q` *`options`* \| `--sql` *`options`* | Configures SQL Data Source access. *`options`* has the following form:<br>`[retry:`*`ms`* `\| fail:`*`n`* `\| db:`*`name`* `\| noinfo \| nopererr \| quote]`<br><br>`retry`: Specify the interval (in milliseconds) to retry connecting to a database after an attempt to connect fails. Default is `-1`, which disables this feature.<br><br>`fail`: Specify the number of consecutive failed SQL queries after which to close this database |

| Option | Description |
| --- | --- |
| | connection and attempt to reconnect. Default is `-1`, which disables this feature.<br><br>`db`: Specify the logical name of the database as specified in the builder's SQL options.<br><br>`noinfo`: Query database for available tables and columns in your database. If a Database Repository file is found, it is used to populate drop down menus in the **Attach to SQL Data** dialog.<br><br>`nopererr`: SQL errors with the word "permission" in them will not be printed to the console. This is helpful if you have selected the **Use Client Credentials** option for a database. In this case, if your login does not allow access for some data in their display, you will not see any errors.<br><br>`quote`: Encloses all table and column names specified in the **Attach to SQL Data** dialog in quotes when an SQL query is run. This is useful when attaching to databases that support quoted case-sensitive table and column names. If a case-sensitive table or column name is used in the **Filter** field, or you are entering an advanced query in the **SQL Query** field, they must be entered in quotes, even if the `-sqlquote` option is specified. |
| `-R` *`bool`* \| `--purgeOnRemove` *`bool`* | Specifies whether to purge all scenario or DataView data when an instance or item is removed. *`bool`* is one of `true` and `false`. If this option is not specified, all scenario or DataView data is purged when an instance or item is removed. |
| `-S` *`variable`*`:`*`value`* \| `--sub` *`variable`*`:`*`value`* | Specifies a value to substitute for a given dashboard variable. This can be used to parameterize a dashboard at startup. This option can occur multiple times in a single command. For example:<br><br>`-S $foo:hello -S $bar:can't -S $tom:"my oh my" -S $jerry:"\"yikes\""`<br><br>If the value contains a space, enclose the value in double quotes. If the value contains a double quote, you must escape it by using a backslash character (\). |
| `-s` *`host`*`:`*`port`*`[:modifiable]` | Specifies the Data Server host and port that will appear as the defaults in the **Data Server Login** |

| Option | Description |
|---|---|
| \| --dashboardServer *host*:*port*[:modifiable] | dialog. The host and port fields of the dialog will be modifiable only if you specify `modifiable`. If you do not specify `modifiable`, the host and port fields will be greyed out. If you use the `-s` option, the **Connect directly to correlator** check box will not appear in the **Data Server Login** dialog. The `-s` option overrides the `-d` option. |
| `-T` *depth* \| `--maxTrend` *depth* | Maximum depth for trend data, that is, the maximum number of events in trend tables. If this option is not specified, the maximum trend depth is 1000. Note that the higher you set this value, the more memory the Data Server requires, and the more time it requires in order to display trend and stock charts. |
| `-t` *value* \| `--title` *value* | Text for the title bar of the Dashboard Viewer main window. |
| `-u` *rate* \| `--updateRate` *rate* | Data update rate in milliseconds. This is the rate at which the Data Server pushes new data to deployed dashboards in order to inform them of new events received from the correlator. *rate* should be no lower than 250. If the Dashboard Viewer is utilizing too much CPU, you can lower the update rate by specifying a higher value. If this option is not specified, an update rate of 500 milliseconds is used. |
| `-V` \| `--version` | Emit program name and version number and then exit. |
| `-v` \| `--loglevel` *level* | Logging verbosity. *level* is one of `FATAL`, `ERROR`, `WARN`, `INFO`, `DEBUG`, and `TRACE`. If this option is not specified, the options in the log4j properties file will be used. |
| `-w` *bool* \| `--disconnectWarning` *bool* | By default, the Dashboard Viewer will display a warning dialog when the connection to a correlator is lost. Specify `false` to disable the display of this dialog. |
| `-X` *file* \| `--extensionFile` *file* | Full pathname of the JAAS initialization file to be used by the Data Server. If not specified, the Data |

| Option | Description |
|---|---|
| | Server uses the file `EXTENSIONS.ini` in the `lib` directory of your Apama installation. |
| `-x` *`table-name`*`:`*`key-list`* \| `--queryIndex` *`table-name`*`:`*`key-list`* | Add an index for the specified SQL-based instance table with the specified compound key. *`table-name`* is the name of a scenario or DataView. *`key-list`* is a comma-separated list of variable names or field names. If the specified scenario or DataView exists in multiple correlators that are connected to the Dashboard Server, the index is added to each corresponding data table. Example:<br>`--queryIndex CrossOver:Instrument,Price`<br>You can only add one index per table, but you can specify this option multiple times in a single command line in order to index multiple tables. Use this option only when you connect the viewer directly to a correlator. |
| `-Y` \| `--enhancedQuery` | Make SQL-based instance tables available as data tables for visualization attachments. See "Attaching Dashboards to Correlator Data" on page 55. Use this option only when you connect the viewer directly to a correlator. |
| `-z` *`zone`* \| `--timezone` *`zone`* | Default time zone for interpreting and displaying dates. *`zone`* is either a Java timezone ID or a custom ID such as `GMT-8:00`. Unrecognized IDs are treated as GMT. See "Timezone ID Values" on page 715 for the complete listing of permissible values for *`zone`*. |
| `--inclusionFilter` *`val`* | Set scenario inclusion filters. This option can occur multiple times in a single command. |
| `--exclusionFilter` *`val`* | Set scenario exclusion filters. This option can occur multiple times in a single command. |

# 28 Timezone ID Values

The following table lists the timezone ID values used when manually starting the Dashboard Viewer as described in "Startup Options for the Dashboard Viewer" on page 709.

| | | |
|---|---|---|
| December 9, 2010 9:40 amtc/GMT+12 Etc/GMT+11 MIT Pacific/Apia Pacific/ Midway Pacific/Niue Pacific/Pago_Pago Pacific/Samoa US/ Samoa America/Adak America/Atka Etc/ GMT+10 HST Pacific/ Fakaofo Pacific/ Honolulu Pacific/ Johnston | PRT SystemV/AST4 SystemV/AST4ADT America/St_Johns CNT Canada/ Newfoundland AGT America/ Araguaina America/ Belem America/ Buenos_Aires America/Catamarca America/Cayenne America/Cordoba America/Fortaleza America/Godthab America/Jujuy | Africa/Khartoum Africa/Mogadishu Africa/Nairobi Antarctica/Syowa Asia/Aden Asia/ Baghdad Asia/Bahrain Asia/Kuwait Asia/ Qatar Asia/Riyadh EAT Etc/GMT-3 Europe/Moscow Indian/Antananarivo Indian/Comoro Indian/Mayotte |
| Pacific/Rarotonga Pacific/Tahiti SystemV/HST10 US/ Aleutian US/Hawaii Pacific/Marquesas AST America/ Anchorage America/ Juneau America/Nome America/Yakutat Etc/GMT+9 Pacific/ Gambier SystemV/ YST9 SystemV/ YST9YDT US/Alaska America/Dawson | America/Maceio America/Mendoza America/Miquelon America/Montevideo America/Paramaribo America/Recife America/Rosario America/Sao_Paulo Antarctica/Rothera BET Brazil/East Etc/ GMT+3 America/ Noronha Atlantic/ South_Georgia Brazil/ DeNoronha Etc/ GMT+2 America/ Scoresbysund | W-SU Asia/Riyadh87 Asia/Riyadh88 Asia/ Riyadh89 Mideast/ Riyadh87 Mideast/ Riyadh88 Mideast/ Riyadh89 Asia/Tehran Iran Asia/Aqtau Asia/Baku Asia/Dubai Asia/Muscat Asia/ Oral Asia/Tbilisi Asia/ Yerevan Etc/GMT-4 |
| America/Ensenada America/Los_Angeles America/Tijuana America/Vancouver America/Whitehorse | Atlantic/Azores Atlantic/Cape_Verde Etc/GMT+1 Africa/ Abidjan Africa/ Accra Africa/Bamako | Europe/Samara Indian/Mahe Indian/ Mauritius Indian/ Reunion NET Asia/ Kabul Asia/Aqtobe |

| | | |
|---|---|---|
| Canada/Pacific Canada/Yukon Etc/ GMT+8 Mexico/ BajaNorte PST PST8PDT Pacific/ Pitcairn SystemV/PST8 SystemV/PST8PDT US/Pacific US/ Pacific-New America/ Boise America/ Cambridge_Bay America/ Chihuahua America/ Dawson_Creek America/Denver America/Edmonton America/Hermosillo America/Inuvik America/Mazatlan America/Phoenix America/Shiprock America/Yellowknife Canada/Mountain Etc/GMT+7 MST MST7MDT Mexico/ BajaSur Navajo PNT SystemV/MST7 SystemV/MST7MDT US/Arizona US/ Mountain America/ Belize America/ Cancun America/ Chicago America/ Costa_Rica America/ El_Salvador America/ Guatemala America/ Managua America/ Menominee America/ Monterrey | Africa/Banjul Africa/ Bissau Africa/ Casablanca Africa/ Conakry Africa/Dakar Africa/El_Aaiun Africa/Freetown Africa/Lome Africa/ Monrovia Africa/ Nouakchott Africa/ Ouagadougou Africa/ Sao_Tome Africa/ Timbuktu America/ Danmarkshavn Atlantic/Canary Atlantic/Faeroe Atlantic/Madeira Atlantic/Reykjavik Atlantic/St_Helena Eire Etc/GMT Etc/ GMT+0 Etc/GMT-0 Etc/GMT0 Etc/ Greenwich Etc/ UCT Etc/UTC Etc/ Universal Etc/Zulu Europe/Belfast Europe/Dublin Europe/Lisbon Europe/London GB GB-Eire GMT GMT0 Greenwich Iceland Portugal UCT UTC | Asia/Ashgabat Asia/Ashkhabad Asia/Bishkek Asia/ Dushanbe Asia/ Karachi Asia/ Samarkand Asia/ Tashkent Asia/ Yekaterinburg Etc/ GMT-5 Indian/ Kerguelen Indian/ Maldives PLT Asia/ Calcutta IST Asia/ Katmandu Antarctica/ Mawson Antarctica/ Vostok Asia/Almaty Asia/Colombo Asia/ Dacca Asia/Dhaka Asia/Novosibirsk Asia/Omsk Asia/ Qyzylorda Asia/ Thimbu Asia/ Thimphu BST Etc/ GMT-6 Indian/Chagos Asia/Rangoon Indian/ Cocos Antarctica/ Davis Asia/Bangkok Asia/Hovd Asia/ Jakarta Asia/ Krasnoyarsk Asia/ Phnom_Penh Asia/ Pontianak Asia/Saigon Asia/Vientiane Etc/ GMT-7 |
| America/ Merida America/ Mexico_City America/ North_Dakota/ Center America/ Rainy_River America/ Rankin_Inlet America/ Regina America/ | Universal WET Zulu Africa/Algiers Africa/Bangui Africa/ Brazzaville Africa/ Ceuta Africa/Douala Africa/Kinshasa Africa/Lagos Africa/ Libreville Africa/ | Indian/Christmas VST Antarctica/ Casey Asia/Brunei Asia/Chongqing Asia/Chungking Asia/Harbin Asia/ Hong_Kong Asia/ Irkutsk Asia/Kashgar |

| | | |
|---|---|---|
| Swift_Current America/Tegucigalpa America/Winnipeg CST CST6CDT Canada/Central Canada/East-Saskatchewan Canada/Saskatchewan Chile/EasterIsland Etc/GMT+6 Mexico/General Pacific/Easter Pacific/Galapagos SystemV/CST6 SystemV/CST6CDT US/Central America/Bogota America/Cayman America/Detroit America/Eirunepe America/Fort_Wayne America/Grand_Turk America/Guayaquil America/Havana America/Indiana/Indianapolis America/Indiana/Knox America/Indiana/Marengo America/Indiana/Vevay America/Indianapolis America/Iqaluit America/Jamaica America/Kentucky/Louisville America/Kentucky/Monticello America/Knox_IN America/Lima America/Louisville America/Montreal America/Nassau America/New_York America/Nipigon America/Panama | Luanda Africa/Malabo Africa/Ndjamena Africa/Niamey Africa/Porto-Novo Africa/Tunis Africa/Windhoek Arctic/Longyearbyen Atlantic/Jan_Mayen CET ECT Etc/GMT-1 Europe/Amsterdam Europe/Andorra Europe/Belgrade Europe/Berlin Europe/Bratislava Europe/Brussels Europe/Budapest Europe/Copenhagen Europe/Gibraltar Europe/Ljubljana Europe/Luxembourg Europe/Madrid Europe/Malta Europe/Monaco Europe/Oslo Europe/Paris Europe/Prague Europe/Rome Europe/San_Marino Europe/Sarajevo Europe/Skopje Europe/Stockholm Europe/Tirane Europe/Vaduz | Asia/Kuala_Lumpur Asia/Kuching Asia/Macao Asia/Macau Asia/Makassar Asia/Manila Asia/Shanghai Asia/Singapore Asia/Taipei Asia/Ujung_Pandang Asia/Ulaanbaatar Asia/Ulan_Bator Asia/Urumqi Australia/Perth Australia/West CTT Etc/GMT-8 Hongkong PRC Singapore Asia/Choibalsan Asia/Dili Asia/Jayapura Asia/Pyongyang Asia/Seoul Asia/Tokyo Asia/Yakutsk Etc/GMT-9 JST Japan Pacific/Palau ROK ACT Australia/Adelaide Australia/Broken_Hill Australia/Darwin Australia/North |
| America/Pangnirtung America/Port-au-Prince America/Porto_Acre America/ | Europe/Vatican Europe/Vienna Europe/Warsaw Europe/Zagreb | Australia/South Australia/Yancowinna AET Antarctica/DumontDUrville |

| | | |
|---|---|---|
| Rio_Branco America/ Thunder_Bay Brazil/ Acre Canada/ Eastern Cuba EST EST5EDT Etc/GMT +5 IET Jamaica SystemV/EST5 SystemV/EST5EDT US/East-Indiana US/Eastern US/ Indiana-Starke US/ Michigan America/ Anguilla America/ Antigua America/ Aruba America/ Asuncion America/ Barbados America/ Boa_Vista America/ Caracas America/ Cuiaba America/ Curacao America/ Dominica America/ Glace_Bay America/ Goose_Bay America/ Grenada America/ Guadeloupe America/ Guyana America/ Halifax America/ La_Paz America/ Manaus America/ Martinique America/ Montserrat America/ Port_of_Spain America/ Porto_Velho America/ Puerto_Rico America/ Santiago America/ Santo_Domingo America/St_Kitts America/St_Lucia America/St_Thomas America/St_Vincent | Europe/Zurich MET Poland ART Africa/ Blantyre Africa/ Bujumbura Africa/ Cairo Africa/Gaborone Africa/Harare Africa/Johannesburg Africa/Kigali Africa/ Lubumbashi Africa/ Lusaka Africa/Maputo Africa/Maseru Africa/ Mbabane Africa/ Tripoli Asia/Amman Asia/Beirut Asia/ Damascus Asia/ Gaza Asia/Istanbul Asia/Jerusalem Asia/ Nicosia Asia/Tel_Aviv CAT EET Egypt Etc/ GMT-2 Europe/Athens Europe/Bucharest Europe/Chisinau Europe/Helsinki Europe/Istanbul Europe/Kaliningrad Europe/Kiev Europe/ Minsk Europe/Nicosia Europe/Riga Europe/ Simferopol Europe/ Sofia Europe/Tallinn Europe/Tiraspol Europe/Uzhgorod | Asia/Sakhalin Asia/ Vladivostok Australia/ ACT Australia/ Brisbane Australia/ Canberra Australia/ Hobart Australia/ Lindeman Australia/ Melbourne Australia/ NSW Australia/ Queensland Australia/ Sydney Australia/ Tasmania Australia/ Victoria Etc/GMT-10 Pacific/Guam Pacific/ Port_Moresby Pacific/ Saipan Pacific/Truk Pacific/Yap Australia/ LHI Australia/ Lord_Howe Asia/ Magadan Etc/GMT-11 Pacific/Efate Pacific/ Guadalcanal Pacific/ Kosrae Pacific/ Noumea Pacific/ Ponape SST Pacific/ Norfolk Antarctica/ McMurdo Antarctica/ South_Pole Asia/ Anadyr Asia/ Kamchatka Etc/ GMT-12 Kwajalein NST NZ Pacific/ Auckland Pacific/ Fiji Pacific/Funafuti Pacific/Kwajalein Pacific/Majuro Pacific/ Nauru |
| America/Thule America/Tortola America/Virgin Antarctica/Palmer Atlantic/Bermuda | Europe/Vilnius Europe/Zaporozhye Israel Libya Turkey Africa/Addis_Ababa Africa/Asmera Africa/ | Pacific/Tarawa Pacific/ Wake Pacific/Wallis NZ-CHAT Pacific/ Chatham Etc/GMT-13 Pacific/Enderbury |

| | | |
|---|---|---|
| Atlantic/Stanley Brazil/West Canada/ Atlantic Chile/ Continental Etc/GMT +4 | Dar_es_Salaam Africa/ Djibouti Africa/ Kampala | Pacific/Tongatapu Etc/GMT-14 Pacific/ Kiritimati |