software AG

Building Dashboards

5.2.0

August 2014

APAMA

This document applies to Apama 5.2.0 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.

Copyright © 2013-2014 Software AG, Darmstadt, Germany and/or Software AG USA Inc., Reston, VA, USA, and/or its Subsidiaries and or/its Affiliates and/or their licensors.

Use, reproduction, transfer, publication or disclosure is prohibited except as specifically provided for in your License Agreement with Software AG.

The name Software AG and all Software AG product names are either trademarks or registered trademarks of Software AG and/or Software AG USA Inc. and/or its Subsidiaries and/or its Affiliates and/or their licensors. Other company and product names mentioned herein may be trademarks of their respective owners.

Detailed information on trademarks and patents owned by Software AG and/or its subsidiaries is located at http://documentation.softwareag.com/legal/.

Use of this software is subject to adherence to Software AG's licensing conditions and terms. These terms are located at http://documentation.softwareag.com/legal/ and/or in the root installation directory of the licensed product(s).

This software may include portions of third-party products. For third-party copyright notices and license terms, please refer to "License Texts, Copyright Notices and Disclaimers of Third Party Products." This document is located at http://documentation.softwareag.com/legal/ and/or in the root installation directory of the licensed product(s).

Document ID: PAM-Building_Dashboards-5.2.0-20140808@233876

Table of Contents

Preface

How This Book Is Organized

The information in this book is organized as follows:

- "Introduction" on page 12 introduces the concepts underlying dashboards.
- "Using Dashboard Builder" on page 18 illustrates how to use the Dashboard Builder's interactive functionality.
- "Attaching Dashboards to Correlator Data" on page 51 describes the correlator data that is available for attachment and it describes the most common objects that can be attached to the data.
- "Using Dashboard Functions" on page 130 covers using builtin and user-defined dashboard functions.
- "Reusing Dashboard Components" on page 162 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.
- "Defining Dashboard Commands" on page 141 details the how to integrate scenario commands into a dashboard to create, edit, and delete scenario instances.
- "Sending Events to Correlators" on page 220 details the how to integrate scenario commands into a dashboard to send arbitrary events to correlators.
- "Using XML Data" on page 228 describes how to augment your dashboard by using XML data files as a data source in addition to Apama scenarios and DataViews.
- "Using SQL Data" on page 234 provides information on accessing JDBC or ODBC enabled databases.

Preface

Documentation roadmap

On Windows platforms, the specific set of documentation provided with Apama depends on whether you choose the Developer, Server, or User installation option. On UNIX platforms, only the Server option is available.

Apama provides documentation in three formats:

- HTML viewable in a Web browser

- PDF
- Eclipse Help (if you select the Apama Developer installation option)

On Windows, to access the documentation, select Start > All Programs > Software AG > Apama 5.2 > Apama Documentation . On UNIX, display the `index.html` file, which is in the `doc` directory of your Apama installation directory.

The following table describes the PDF documents that are available when you install the Apama Developer option. A subset of these documents is provided with the Server and User options.

Title	Contents
What's New in Apama	Describes new features and changes since the previous release.
Installing Apama	Instructions for installing the Developer, Server, or User Apama installation options.
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 Studio	Instructions for using Apama Studio to create and test Apama projects; write, profile, and debug EPL programs; write JMon programs; develop custom blocks; and store, retrieve and playback data.
Developing Apama Applications in Event Modeler	Instructions for using Apama Studio's Event Modeler editor to develop scenarios. Includes information about using standard functions, standard blocks, and blocks generated from scenarios.
Developing Apama Applications in EPL	Introduces Apama's Event Processing Language (EPL) and provides user guide type information for how to write EPL programs. EPL is the native interface to the correlator. This document also provides information for using the standard correlator plug-ins.
Apama EPL Reference	Reference information for EPL: lexical elements, syntax, types, variables, event definitions, expressions, statements.
Developing Apama Applications in Java	Introduces the Apama in-process API for Java, referred to as JMon, and provides user guide type information for how to write Java programs that run on the correlator. Reference information in Javadoc format is also available.
Building Dashboards	Describes how to create dashboards, which are the end-user interfaces to running scenario instances and data view items.
Dashboard Property Reference	Reference information on the properties of the visualization objects that you can include in your dashboards.

Title	Contents
Dashboard Function Reference	Reference information on dashboard functions, which allow you to operate on correlator data before you attach it to visualization objects.
Developing Adapters	Describes how to create adapters, which are components that translate events from non-Apama format to Apama format.
Developing Clients	Describes how to develop C, C++, Java, or .NET clients that can communicate with and interact with the correlator.
Writing Correlator Plug-ins	Describes how to develop formatted libraries of C, C++ or Java functions that can be called from EPL.
Deploying and Managing Apama Applications	Describes how to: • Use the Management & Monitoring console to configure, start, stop, and monitor the correlator and adapters across multiple hosts. • Deploy dashboards over wide area networks, including the internet, and provide dashboards with effective authorization and authentication. • Improve Apama application performance by using multiple correlators, and saving and reusing a snapshot of a correlator's state. • Use the Apama ADBC adapter to store and retrieve data in JDBC, ODBC, and Apama Sim databases. • Use the Apama Web Services Client adapter to invoke Web Services. • Use correlator-integrated messaging for JMS to reliably send and receive JMS messages in Apama applications. • Use Universal Messaging to connect correlators.
Using the Dashboard Viewer	Describes how to view and interact with dashboards that are receiving run-time data from the correlator.

Preface

Contacting customer support

You may open Apama Support Incidents online via the eService section of Empower at http://empower.softwareag.com. If you are new to Empower, send an email to `empower@softwareag.com` with your name, company, and company email address to request an account.

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

Preface

Chapter 1: Introduction

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

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

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.

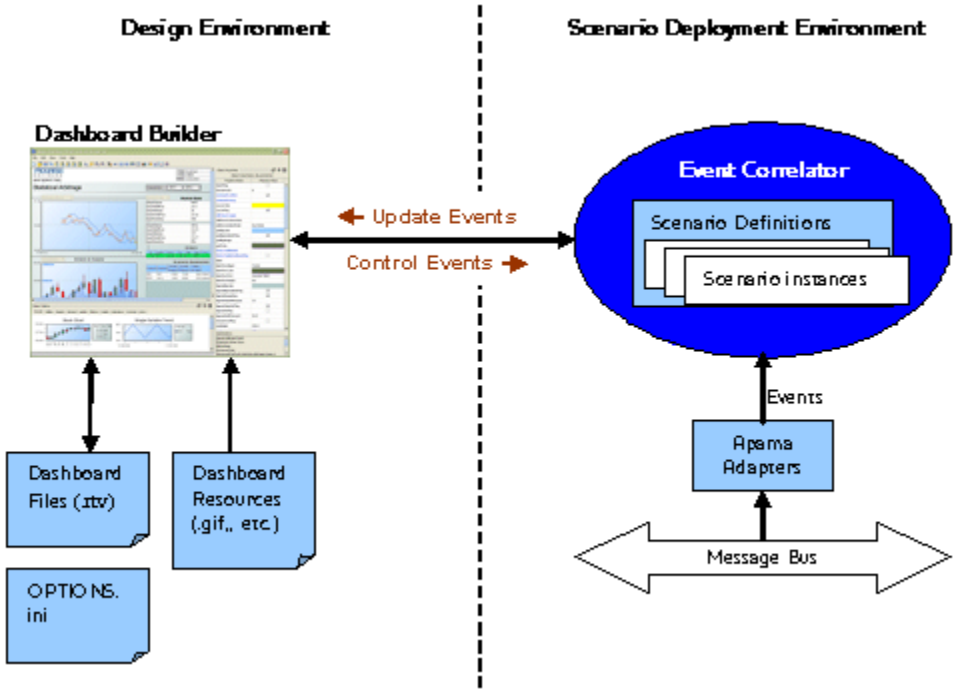
An Apama project typically uses of one or more dashboards. Each dashboard defines a single display, or view, of information. Dashboards are created in the Dashboard Builder (or with the Dashboard Generation Wizard—see *Using Apama Studio*). Each dashboard is stored in a separate `.rtv` file. All `.rtv` 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:

Figure 1. Dashboard-correlator communication in the development environment



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.

Introduction

Starting the Dashboard Builder

You can start the Dashboard Builder from the Windows Start menu, from Apama Studio, or from the command line.

Introduction

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 > Apama 5.2 > Development > Dashboard Builder .

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

Starting the Dashboard Builder

Starting Builder from Apama Studio

You can use Apama Studio to open a specified file in the Builder.

1. Do one of the following:
   - Double-click on a dashboard file in the navigation pane.
   - Right-click on 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.

Starting the Dashboard Builder

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" on page 46.

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

   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 Apama Studio

Starting Builder from the command line

Dashboard Builder can be started by running `dashboard_builder.exe` 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" on page 46.

To run the Builder from the command line:

1. Do one of the following:

- Use an Apama command prompt (select Start > All Programs > Software AG > Apama 5.2 > Apama Command Prompt)
- Set your current directory to the Apama `bin` directory.

Starting the Dashboard Builder

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. See *Deploying and Managing Apama Applications* for information on customizing access control.

Introduction

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.

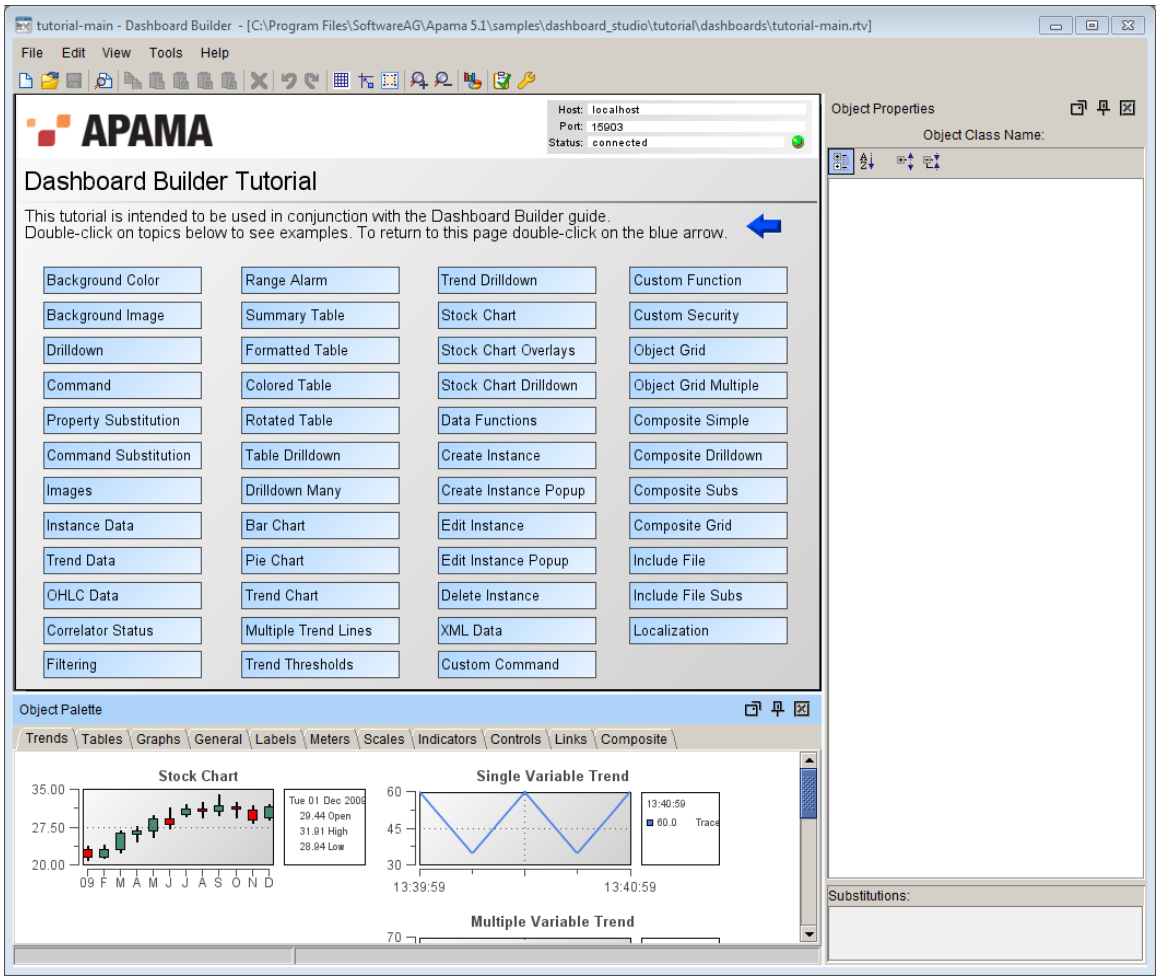
The Dashboard Builder connects to one or more correlators in order to discover the scenarios and DataViews that the correlators have loaded. Information about these scenarios and DataViews is then made available for use in the design of the dashboard.

Follow these steps to run the tutorial:

1. In the Apama Studio Workbench perspective, from the menu bar, 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.

Apama Studio 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.

Figure 2. Tutorial main page in Dashboard Builder



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.

You can run the tutorial application and view the tutorial examples with the Dashboard Viewer instead of the Dashboard Builder by running the script `viewDemo.bat` instead of `editDemo.bat`. Follow steps 1-3 above, and then enter the following: `viewDemo`.

Introduction

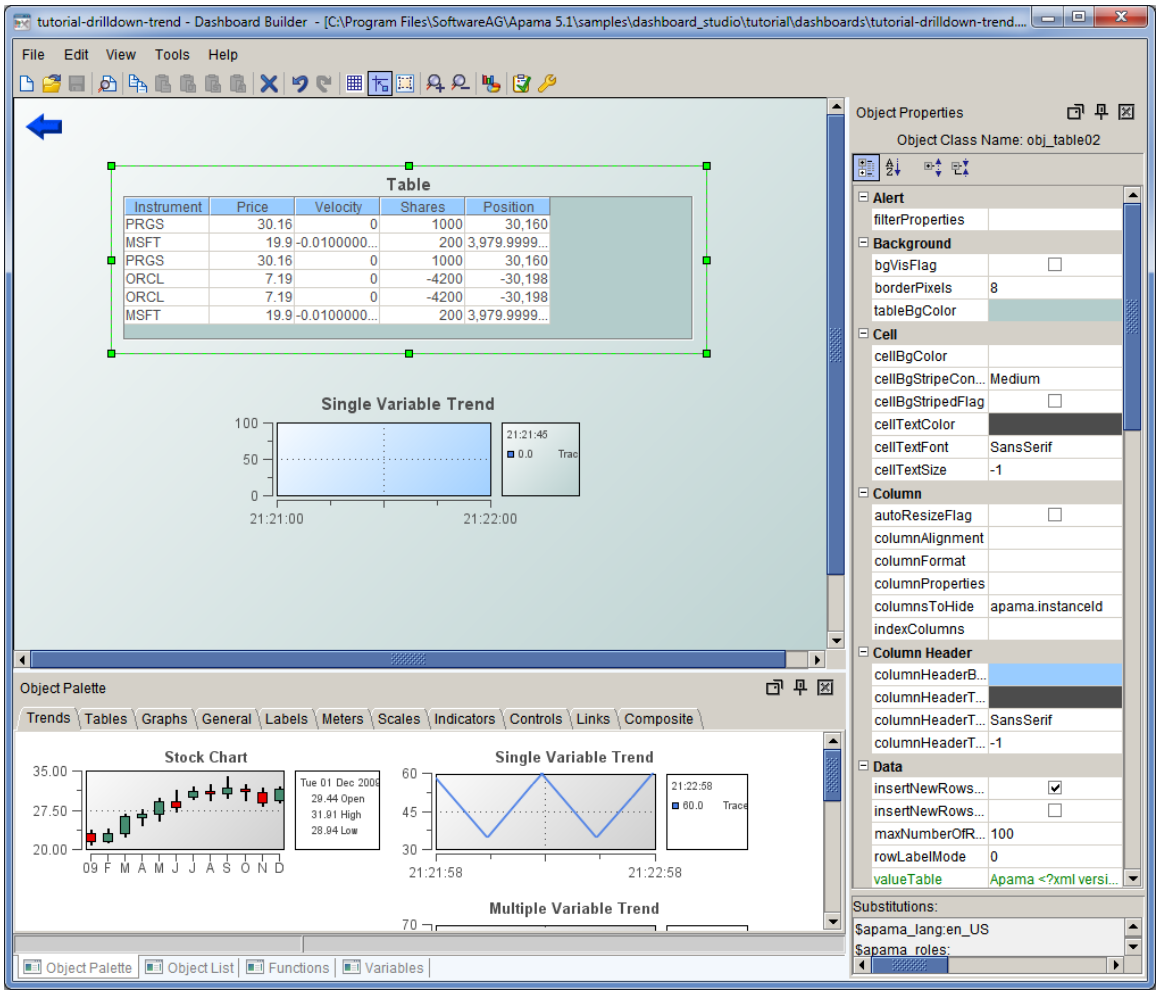
Chapter 2: Using Dashboard Builder

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

The following illustration shows the normal working layout of the Dashboard Builder.

Figure 3. Dashboard Builder layout



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

Using Dashboard Builder

The menubar

There are five menus on the menu bar:

Table 1. Dashboard Builder 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.

Menu	Command	Description
	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.
	Undo	Un-does the last Builder operation (that has no 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

Menu	Command	Description
		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.
	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 on 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 on 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.
	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

Menu	Command	Description
		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 Dashboards*.
	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.

Dashboard Builder layout

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:

Table 2. Dashboard Builder toolbar

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.
	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.

Button	Description
	Display the Object Properties panel.
	Display the Application Options dialog.

Dashboard Builder layout

The canvas

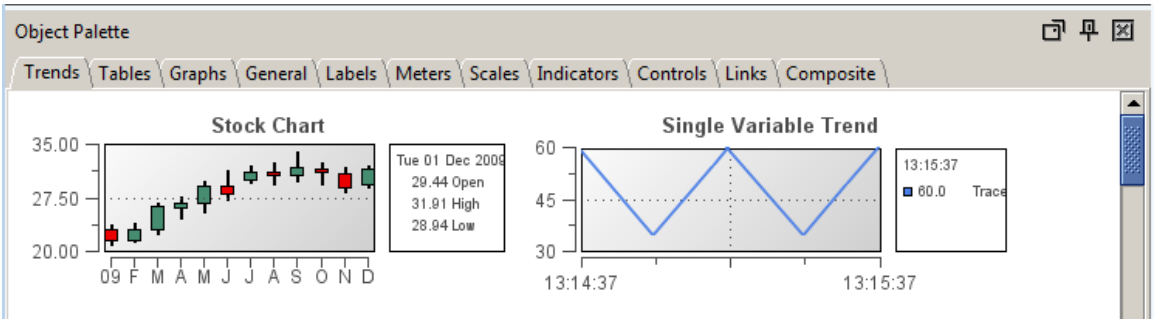
The canvas is where you lay out the objects for a dashboard. Objects can be added to the canvas, moved, and resized. As objects are attached to data sources, the objects will update in real time to reflect changes in the data. This allows you to see how the objects will appear when the dashboard is deployed.

Dashboard Builder layout

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.

Figure 4. Object Palette

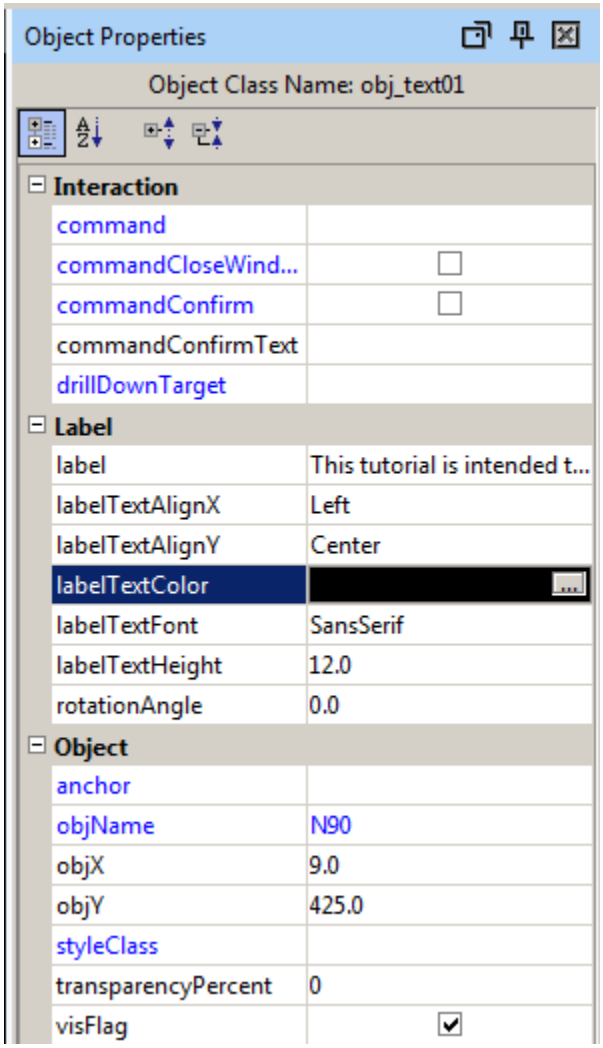


Dashboard Builder layout

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.

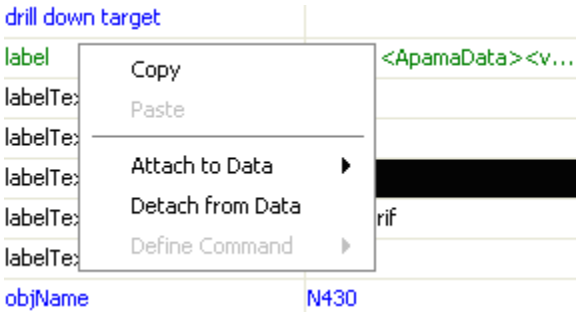
Figure 5. Object Properties panel



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, left click on 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 on 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.

Dashboard Builder layout

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 following sections detail how to define data sources in the Builder.

See also "Using SQL Data" on page 234 for information on JDBC and ODBC data sources.

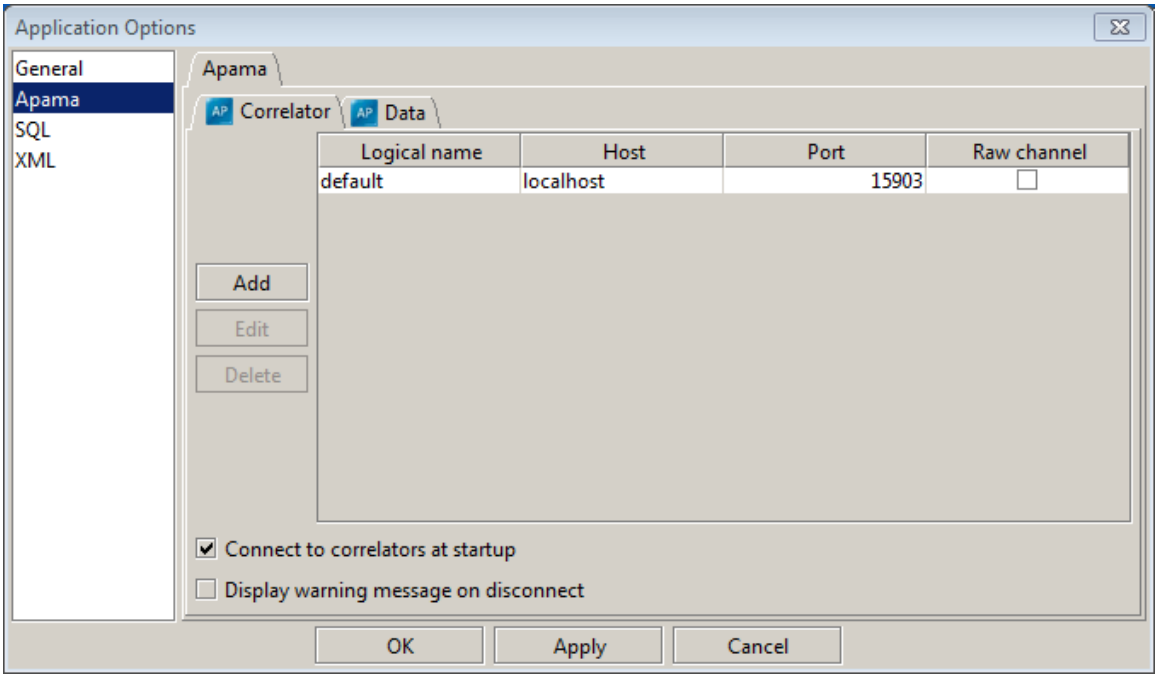
Using Dashboard Builder

Specifying correlators

To create a dashboard for a scenario or DataView in the Dashboard Builder, you first need to specify the correlator in which the scenario or DataView is running.

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

   The Applications Options dialog is shown below.



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 on 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 data sources

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 228, for details on using XML data sources.

Specifying data sources

Activating data source specifications

To activate data source specifications:

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

Specifying data sources

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 46.

Specifying data sources

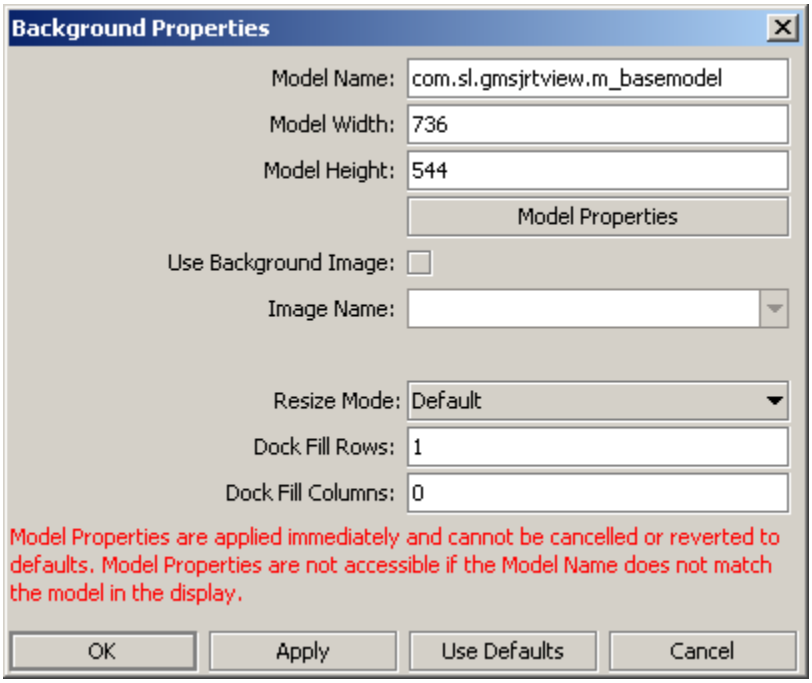
Setting the background properties

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.

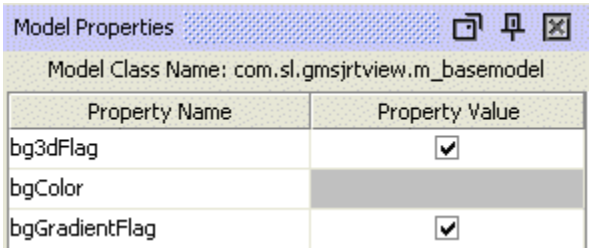
   The **Background Properties** dialog appears.



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.

   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 30.

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 30 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 30 for detailed information.

Using Dashboard Builder

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.

Using Dashboard Builder

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

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.

About resize modes

Working with objects

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

Using Dashboard Builder

Adding objects to a dashboard

Add an object to a dashboard as follows:

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.

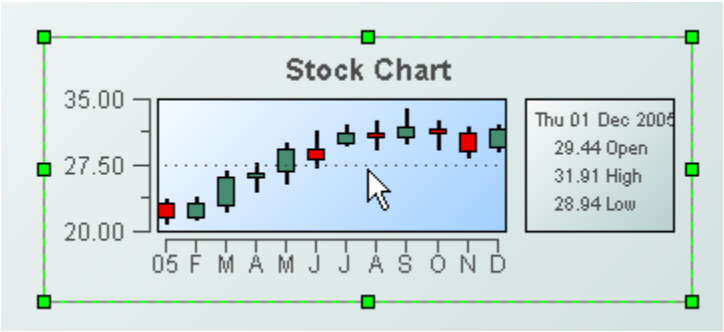
The appearance of the crosshairs pointer indicates that Builder is in add mode, and clicking will add an object to the canvas.

You can adjust the position and size of the object after you have added it.

Working with objects

Selecting an object

Left click on 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.

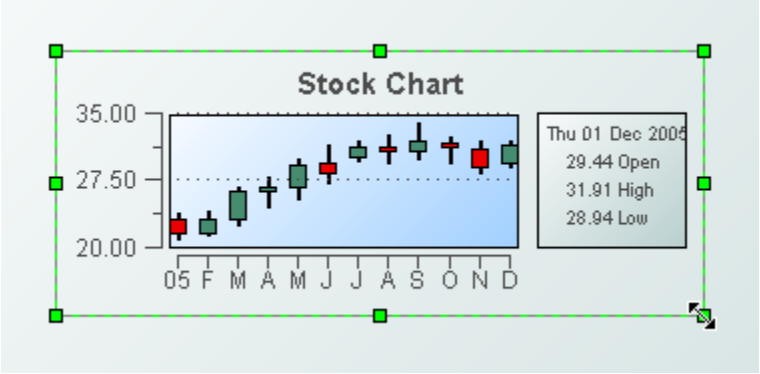
To select multiple objects hold down the Shift key while clicking on the objects.

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

Working with objects

Resizing objects

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

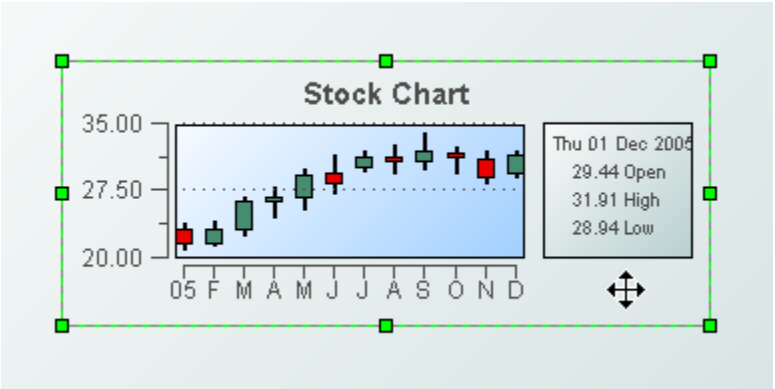


You can also set the size of an object by editing the `objWidth` and `objHeight` properties.

Working with objects

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.

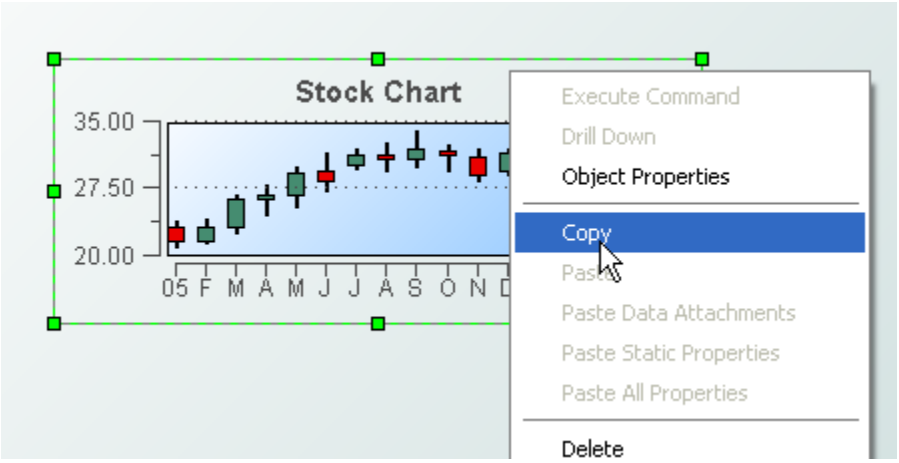
The dashboard canvas uses a Cartesian coordinate system, with the origin (0, 0) in the bottom left corner of the dashboard. The `objX` and `objY` properties are relative to the origin.

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.

Working with objects

Copy and pasting objects

To copy an object, right click on 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.

Working with objects

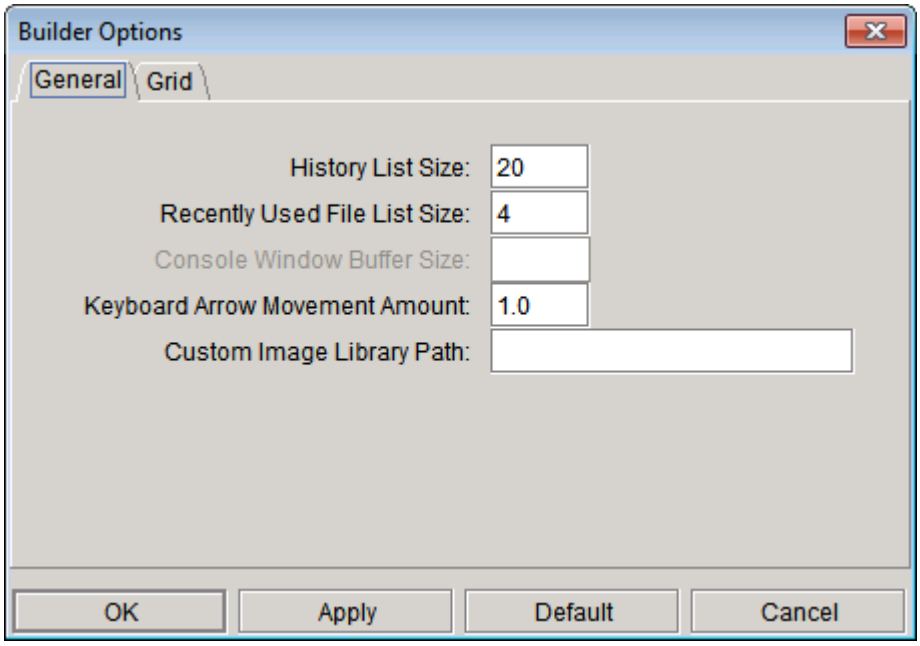
Deleting objects

To delete an object, right click on it and then select Delete from the popup menu. You can also click on 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.

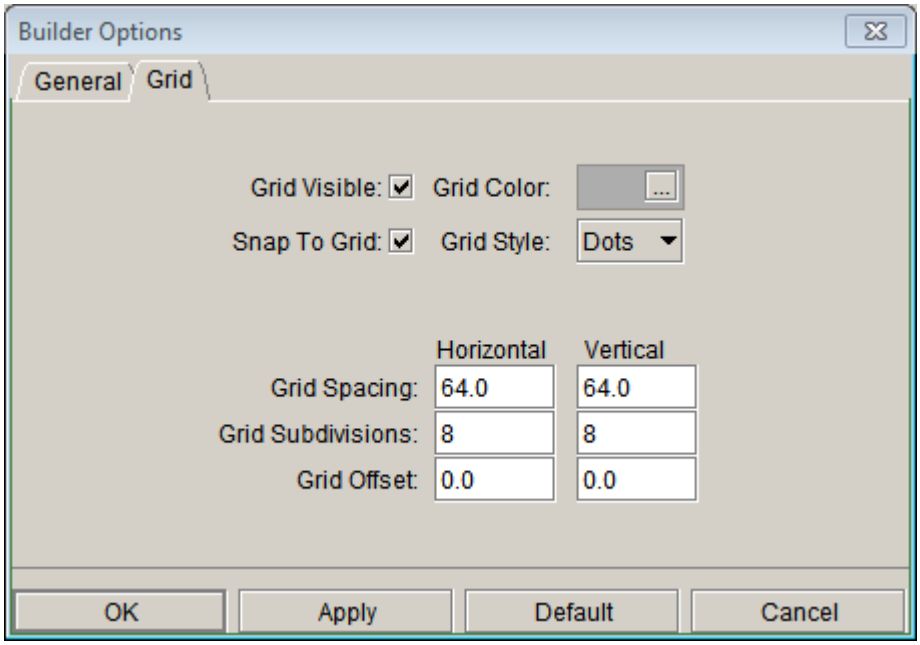
Working with objects

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.



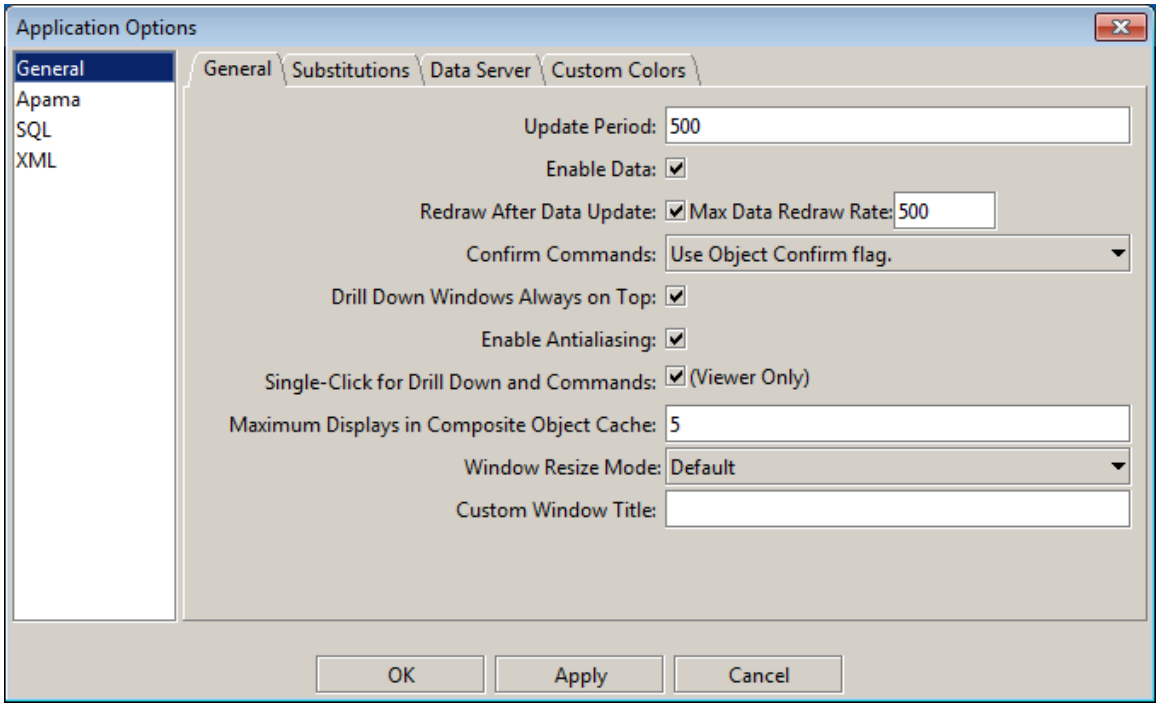
The values set in this dialog are automatically restored on application startup and saved on application exit.

Using Dashboard Builder

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.



The dialog organizes options into tab groups, which are described in the following sections:

- "Setting options in the General tab group" on page 38
- "Setting options in the Apama tab group" on page 44
- "Setting options in the SQL tab group" on page 45
- "Setting options in the XML tab group" on page 45

See also "Saving options" on page 46.

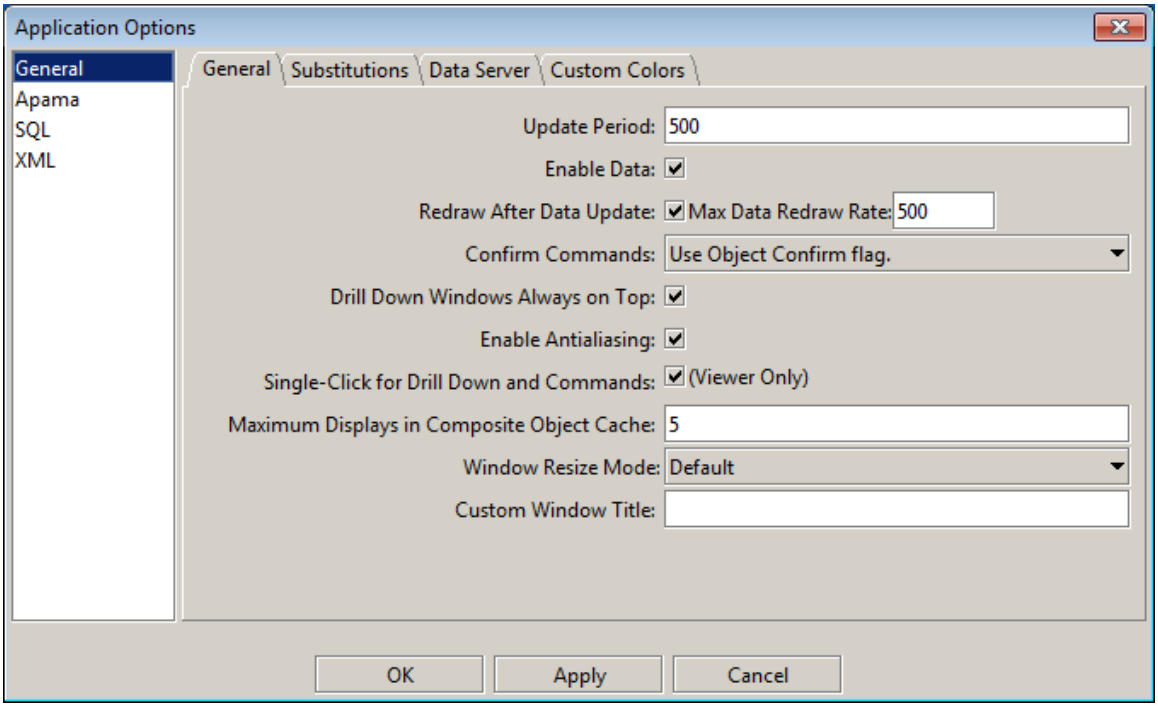
Using Dashboard Builder

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 is displayed.



This tab group has four tabs, which are described in the following sections:

- "Setting options in the General tab" on page 39
- "Setting options in the Substitutions tab" on page 40
- "Setting options in the Data Server tab" on page 40
- "Setting options in the Custom Colors tab" on page 41

Setting Dashboard options

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.

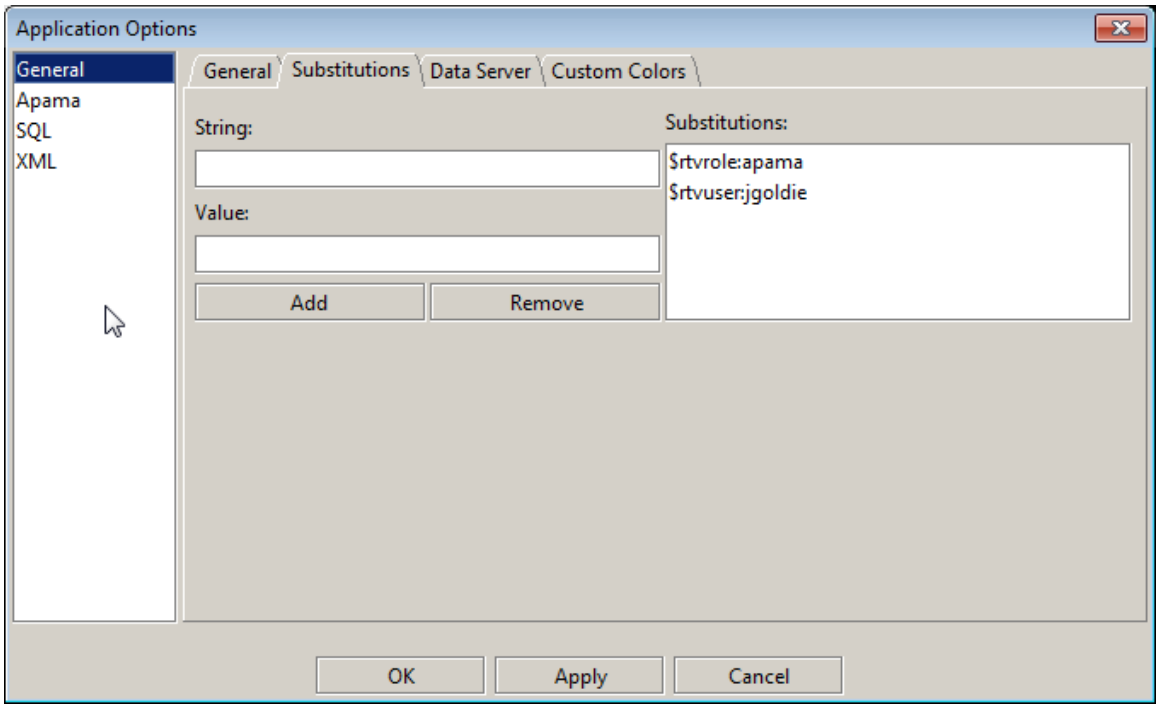
   Data from an asynchronous data source can arrive at any time between update periods. This means there could be a delay between the time an asynchronous data source receives a data update and when the display showing this data is updated. If selected, displays containing data from asynchronous data sources 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. In the Dashboard Builder a double click is always required.
9. In the Maximum Displays in Composite Object Cache field, enter the display caching for composite objects.

Setting options in the General tab group

Setting options in the Substitutions tab

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



Setting options in the General tab group

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.)

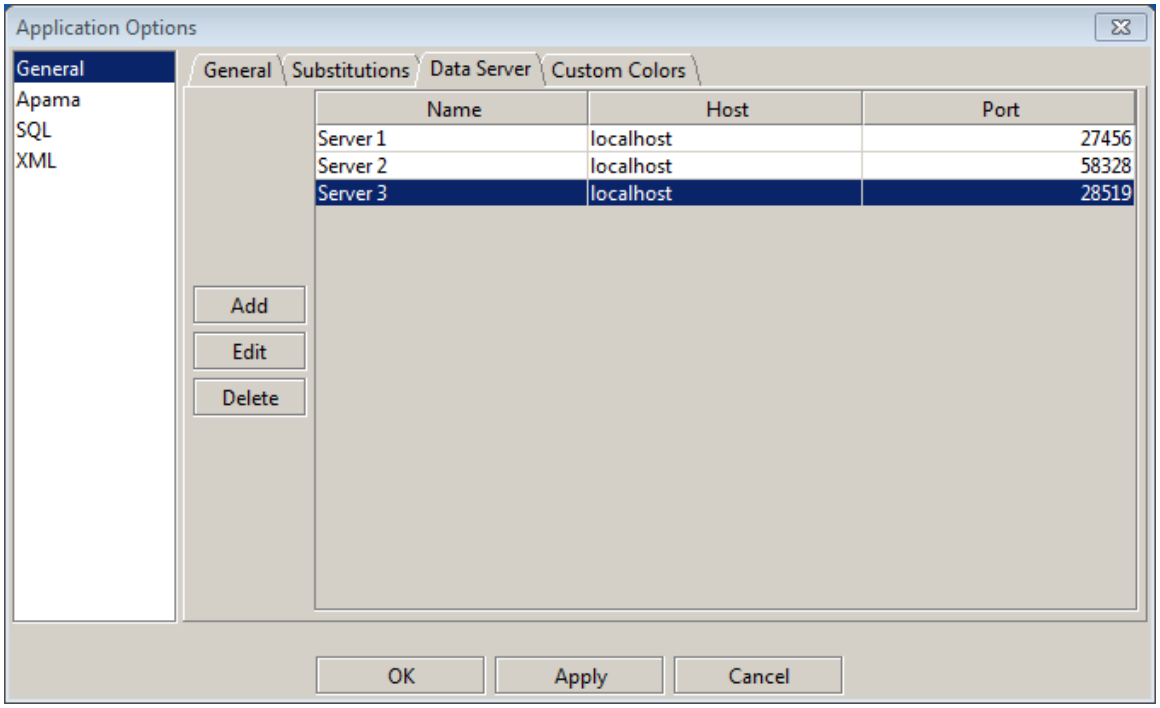
The logical names defined in this tab are used by default for live dashboards viewed with Builder as well as for deplolyed dashboards. They can be overriden with the `--namedDataServer` option to the builder, viewer, or server executables. See "Working with multiple Data Servers" on page 70 for more information.

Follow these steps to define Data Server logical names:

1. Select Options from the Tools menu.

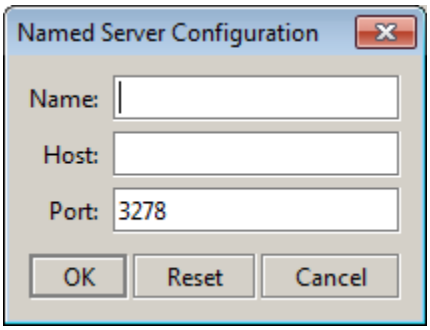
The **Applications Options** dialog box displays.

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 General tab group

Setting options in the Custom Colors tab

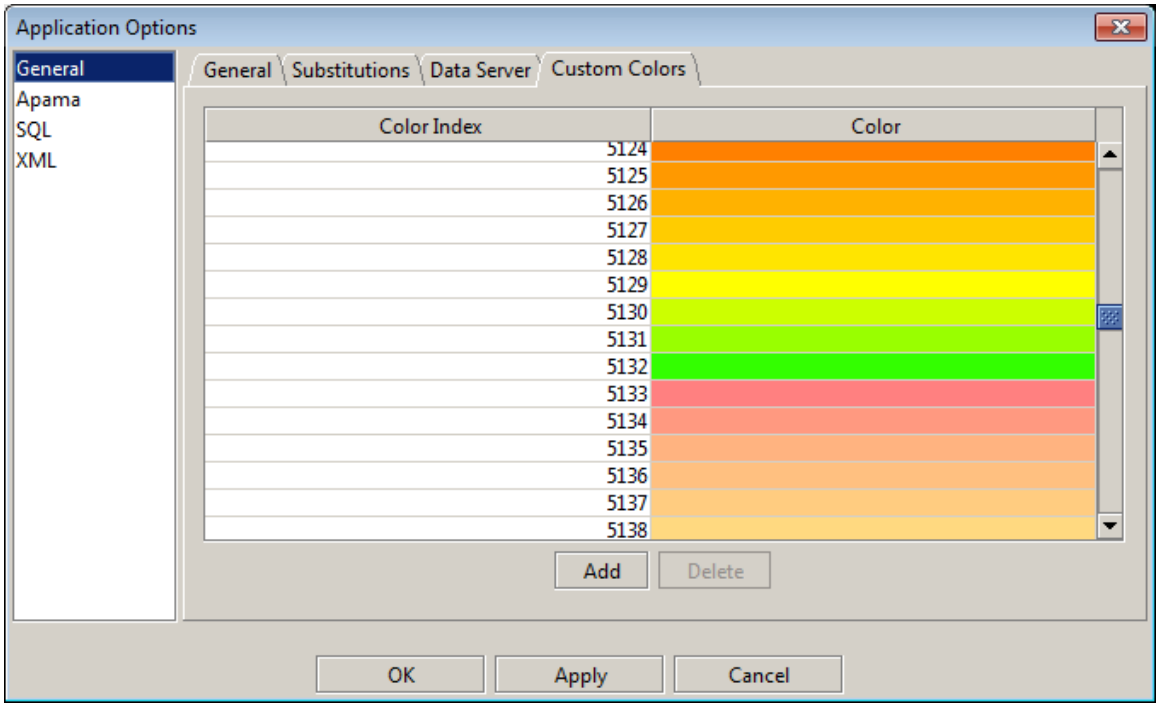
The Custom Colors tab allows you to specify custom colors that you can use to set object property values. (You set color-valued object properties with the Color Chooser window, which has a Standard

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.

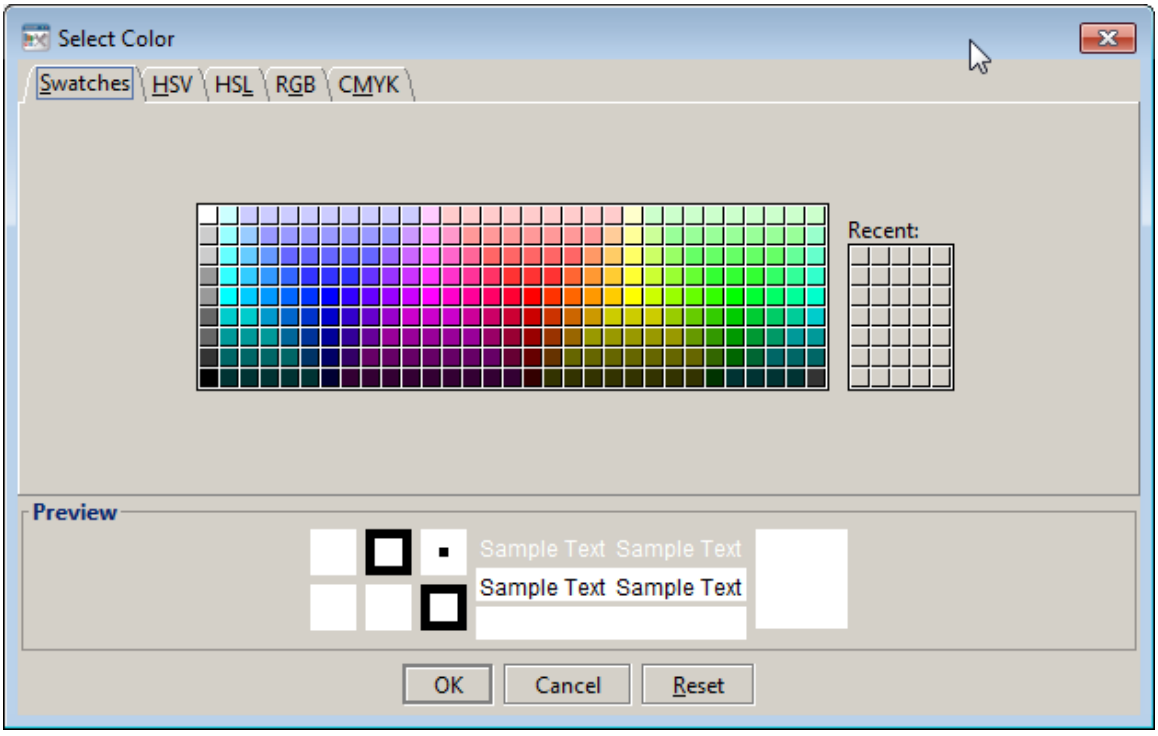
   The **Applications Options** dialog box is displayed.

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 choce by hue, saturation, value, and transparency
   - HSL: Select color choce 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 defition, in the Color fields click on the ... button of a selected color to edit that color definition with the Select Color dialog.

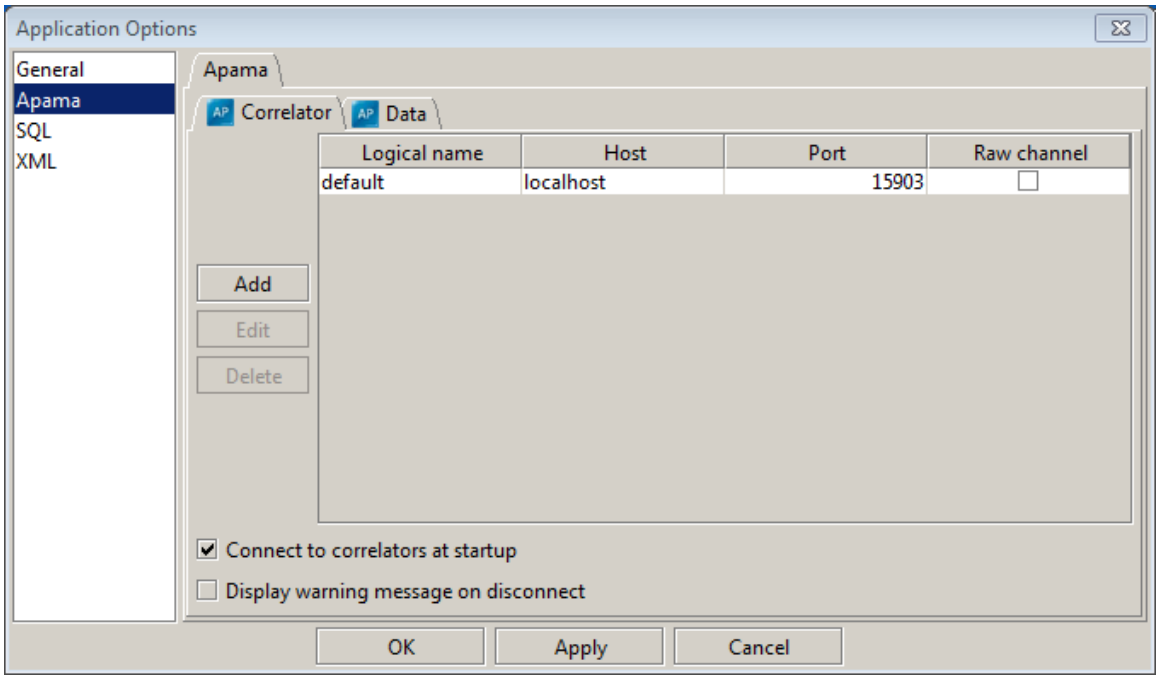
Object limitations: Some objects (for example, the bar graph legend, pie wedges and legend, and some control objects) cache their colors and therefore do not update when a custom color definition changes. To see the color change for these objects, restart Builder or reload the display.

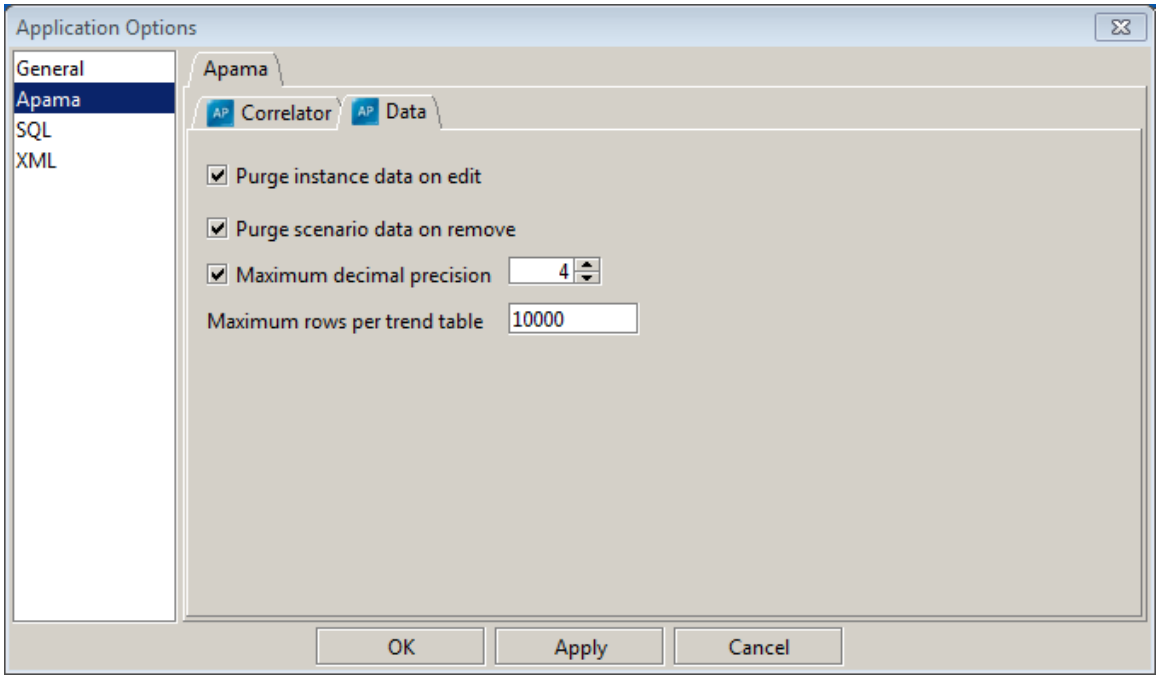
Deployment limitation: Multiple applets running in the same VM share a single Custom Color tab.

Setting options in the General tab group

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 26.



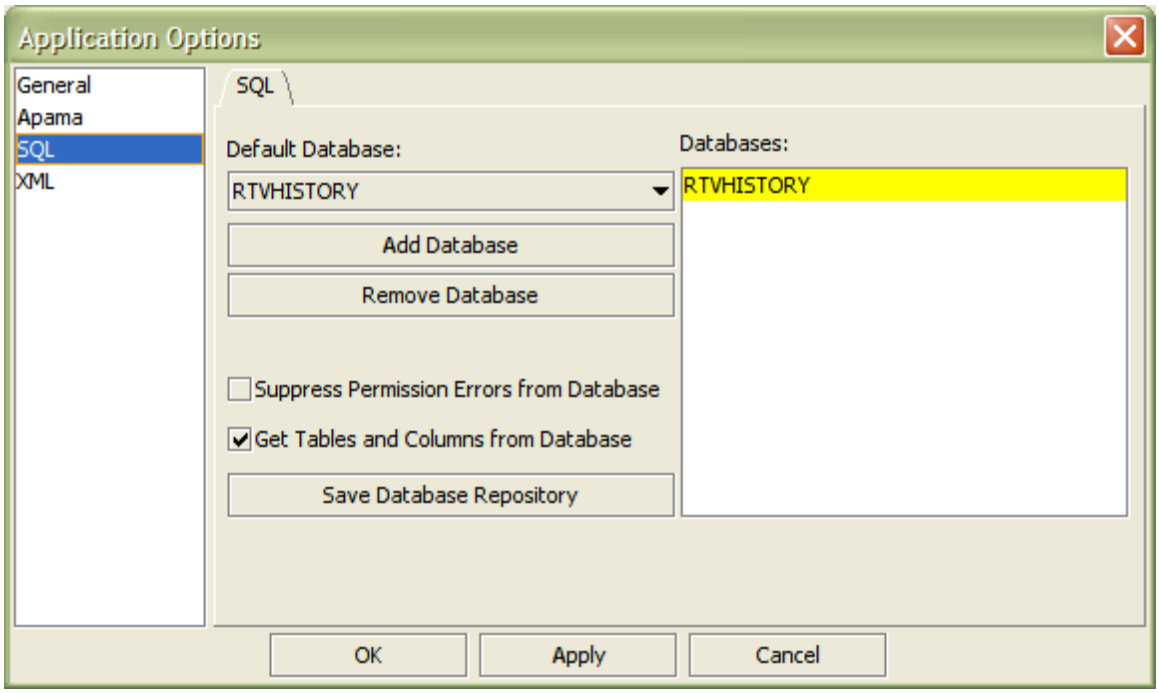


For information on the Data sub tab, see "Specifying data sources" on page 26.

Setting Dashboard options

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 .

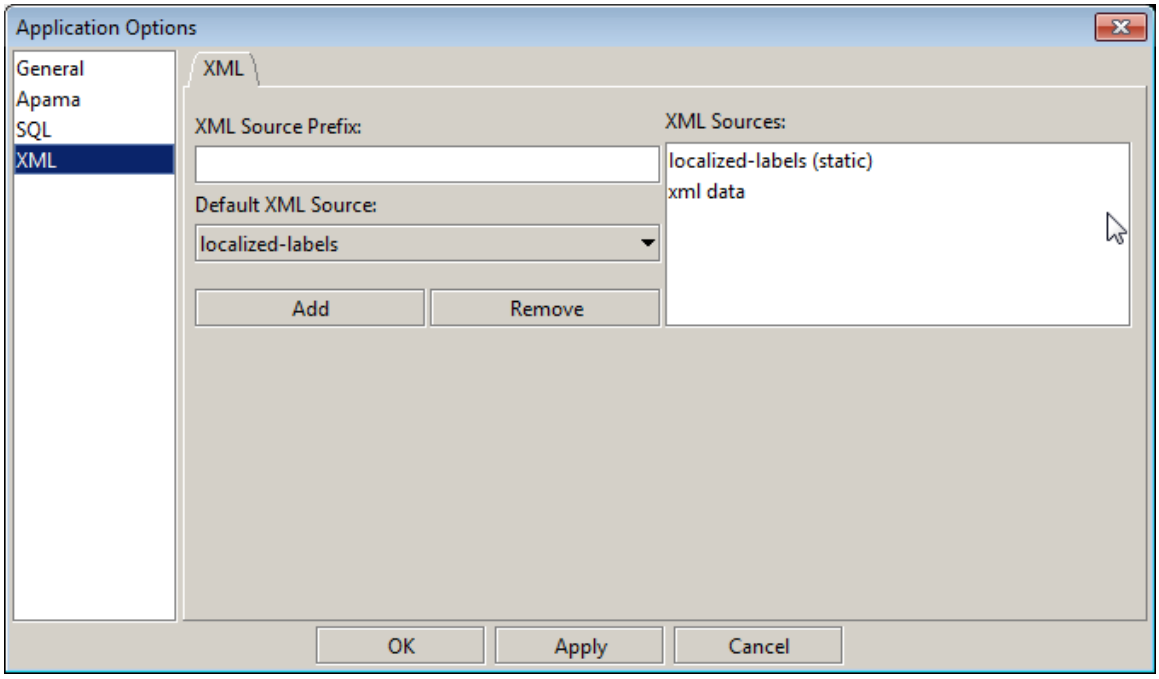


For more information on setting SQL options, see "Specifying application options" on page 240.

Setting Dashboard options

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 228.

Setting Dashboard options

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 instalation (which contains your Apama installation's initial set of custom colors).

Setting Dashboard options

Command line options

The Dashboard Builder executable supports options that can be specified on the start-up command line to override the default values used by the Builder. This section documents these options.

Synopsis

The executable for the Dashboard Builder is `dashboard_builder.exe`. It has the following syntax:

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

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

Options

Following are the command line options for this executable:

Table 3. Dashboard Builder command line options

Option	Description
`-B | --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 70 for more information.

`-c | --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 834 in *Using Studio*.This option can occur multiple times in a single command. For example:
	`-c default:localhost:15903:false`
	`-c work1:somehost:19999:true`
	These options set the host and port for the logical names `default` and `work1`.

`-D | --dashboard directory`	Start with the dashboard found in the specified directory.
`-E | --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 | --filterInstance arg`	Filter scenario instances. This is ignored for Dashboard Server Viewer. Values can be `true` or `false`.
`-f | --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.

Option	Description
`-G | --trendConfigFile` *file*	Trend configuration file for controlling trend-data caching.
`-h | --help`	Emit usage information and then exit.
`-J | --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 | --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 | --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*	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 | --optionsFile` *file*	Use the specified `OPTIONS.ini` file at startup.
`-P | --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 | --sql` *options*	Configures SQL Data Source access. *options* has the following form: `[retry:`*ms* `| fail:`*n* `| db:`*name* `| 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.`db`: Name of SQL database. Only databases using ODBC drivers can be added on the command line`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

Option	Description
	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.
`-R | --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 | --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: `-S $foo:hello -S $bar:can't -S $tom:"my oh my" -S $jerry:"\"yikes\""` 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 | --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 | --title` *value*	Text for the title bar of the Dashboard Builder main window.
`-u | --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.

Option	Description
`-w | --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 | --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.
`-x | --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: `--queryIndex 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.
`-Y | --enhancedQuery`	Make SQL-based instance tables available as data tables for visualization attachments. See Attaching Dashboards to Correlator Data in *Building Dashboards*.
`-z | --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 Appendix A of the *Dashboard Viewer* guide 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.

Using Dashboard Builder

Chapter 3: Attaching Dashboards to Correlator Data

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.

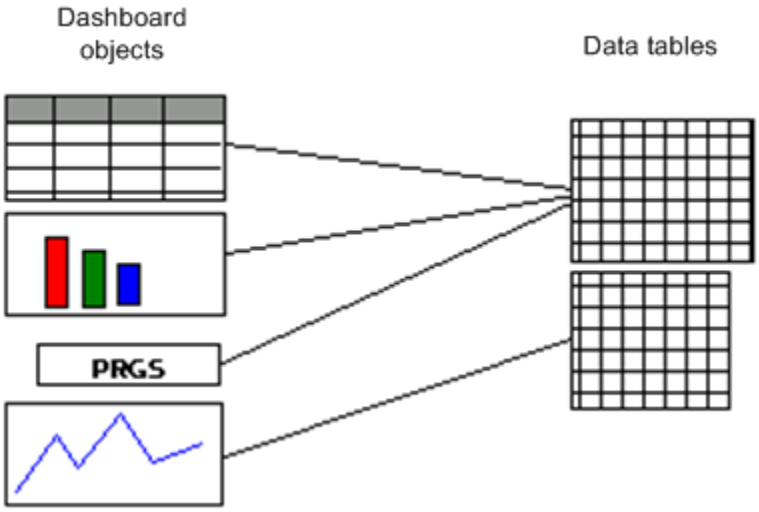
You can attach visualization objects to two kinds of correlator data: scenario data and DataView data. Scenarios and DataViews are described in *Introduction to Apama*.

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 15). 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.

The following sections describe the different types of data tables:

"Scenario instance table" on page 53

"Scenario trend table" on page 54

"Scenario OHLC table" on page 54

"Correlator status table" on page 55

"Data Server status table" on page 55

"Scenario constraint table" on page 56

"DataView item table" on page 56

"DataView trend table" on page 57

"DataView OHLC table" on page 57

"SQL-based instance table" on page 57

"Setting data options" on page 58 provides information on managing data tables.

Attaching Dashboards to Correlator Data

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 following diagram 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 actual 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.

Dashboard data tables

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.

Dashboard data tables

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.

Dashboard data tables

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.

Dashboard data tables

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 following table 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 59. To attach a property to a Data Server status table, attach the property to function data—see "Using Dashboard Functions" on page 130—and specify a function of type Get Data Server Connection Status.)

Dashboard data tables

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 following diagram illustrates the contents of a scenario constraint table.

Parameter	Case	Choices	Contant	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
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 64 for more information on using constraint tables.

Dashboard data tables

DataView item table

A DataView item data table is similar to a scenario instance table (see "Scenario instance table" on page 53). 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.

Dashboard data tables

DataView trend table

A DataView trend data table is similar to a scenario trend table (see "Scenario trend table" on page 54). 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.

Dashboard data tables

DataView OHLC table

A DataView OHLC table is similar to a scenario OHLC table (see "Scenario OHLC table" on page 54). 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.

Dashboard data tables

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 53) and a DataView item table (see "DataView item table" on page 56). 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 67 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 authorization in Administering dashboard security in *Deploying and Managing Apama Applications*). 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 67 for more information.

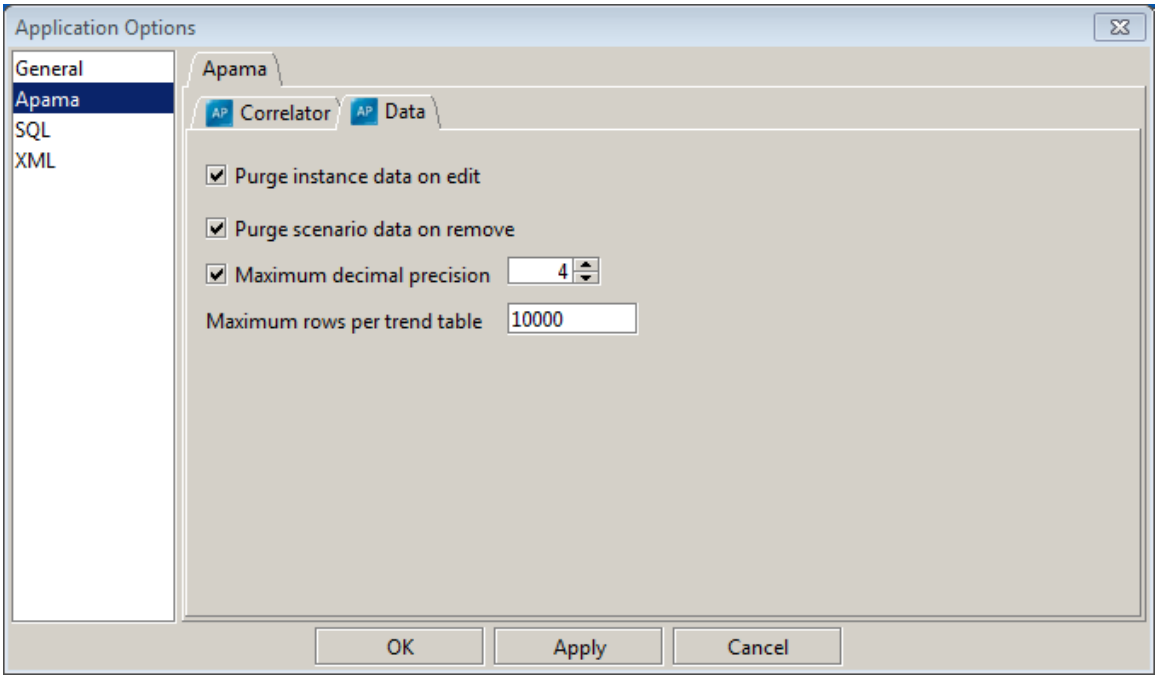
Dashboard data tables

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.

Dashboard data tables

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 67 form more information on SQL query attachments.

See *Deploying and Managing Apama Applications* for information on customizing access control.

Attaching Dashboards to Correlator Data

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.

The following sections cover fundamental tasks and concepts related to creating a data attachment:

"Using the Attach to Apama dialog" on page 60

"Selecting display variables or fields" on page 63

"Displaying attached data" on page 63

"Filtering data" on page 64

"Attaching to constraint data" on page 64

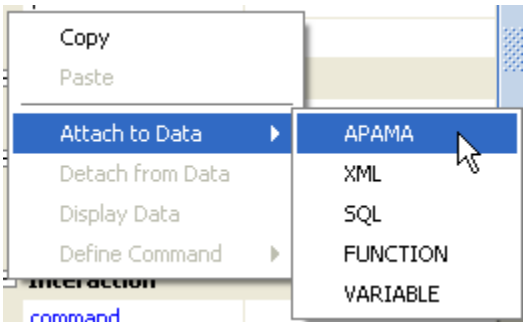
"About timestamps" on page 64
"Using dashboard variables in attachments" on page 65
"About drilldown and $instanceId" on page 66
"About other predefined substitution variables" on page 67
"Using SQL-based instance tables" on page 67
"Working with multiple Data Servers" on page 70
Attaching Dashboards to Correlator Data

Using the Attach to Apama dialog

To attach an object property to Apama data:

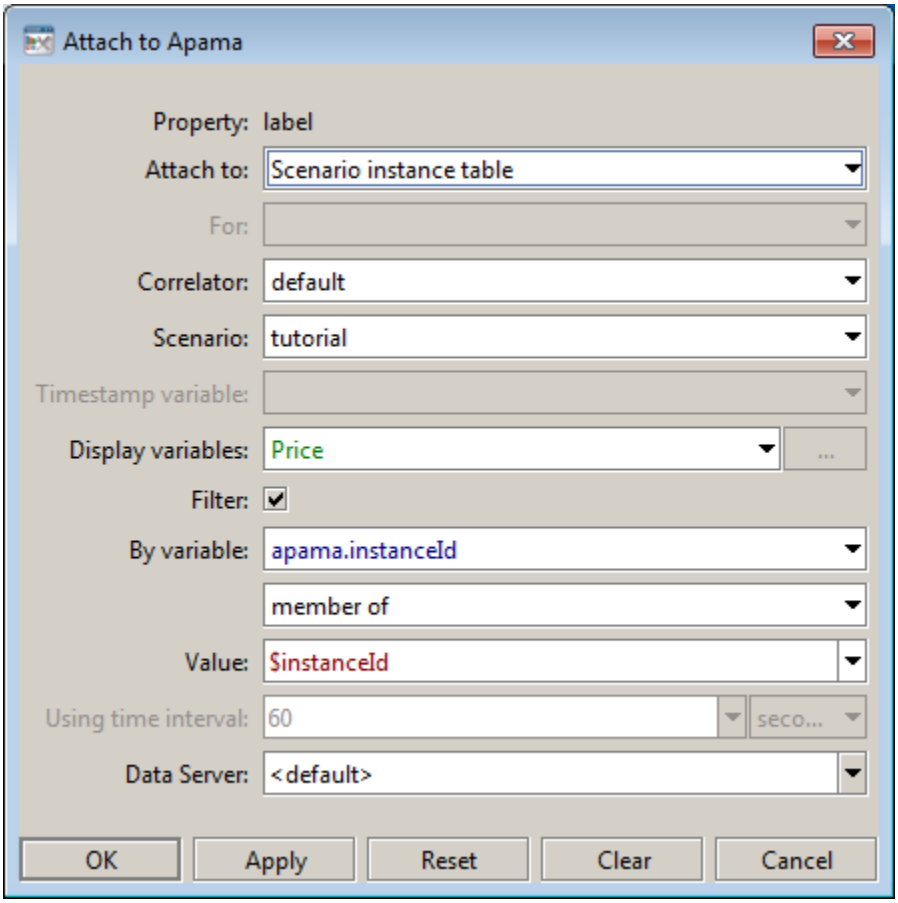
1. Select the property in the property panel and right click it.

   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 67.

To attach a property to a Data Server status table, attach the property to function data—see "Using Dashboard Functions" on page 130—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 64 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 70 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.

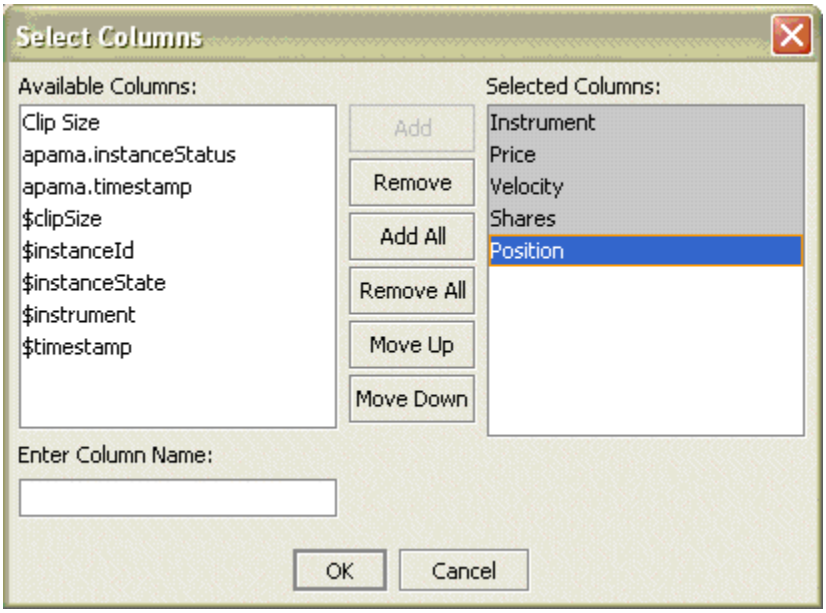
Creating a data attachment

Selecting display variables or fields

Individual display variables or fields can be selected directly in the **Attach to Apama** dialog. If you need to select multiple display variables or fields:

1. Click on the "…" button 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.

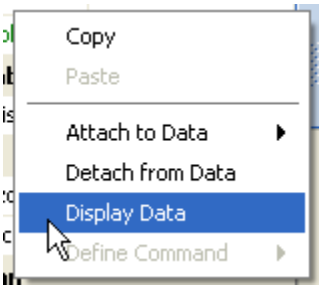
Creating a data attachment

Displaying attached data

Builder provides a convenient way for you to view the data that is currently attached to a given property.

1. Right-click on 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.

Creating a data attachment

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 60 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.

Creating a data attachment

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.

Creating a data attachment

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.

Creating a data attachment

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 66 and "About other predefined substitution variables" on page 67.

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.

Creating a data attachment

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 unchecking the 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 142)
- `-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 169.

Using dashboard variables in attachments

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 `$instanceId`. This variable is automatically set to the value of `apama.instanceId` (see "Scenario instance table" on page 53) for the table row that is currently selected (for tables attached to a scenario instance table). If multiple rows are selected, `$instanceId` is set to multiple values.

For more information and examples, see "Performing drilldowns on tables" on page 81 and "Specifying drill-down column substitutions" on page 84.

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.

Creating a data attachment

About other predefined substitution variables

In addition to `$instanceId` (see "About drilldown and $instanceId" on page 66), Dashboard Builder defines the following substitution variables:

- `$apama_lang`: by default, this variable is set to what Java reports as the locale in the `Locale` object as derived from the host system's locale. You can allow end users to set this to their required locale, and use it to localize dashboard labels. See "Localizing Dashboard Labels" on page 125.
- `$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.
- `$apama_timestamp`: by default, this variable is set to the value of `apama.timestamp` of the scenario instance that is currently selected. See "About timestamps" on page 64.
- `$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.

Creating a data attachment

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 57 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.

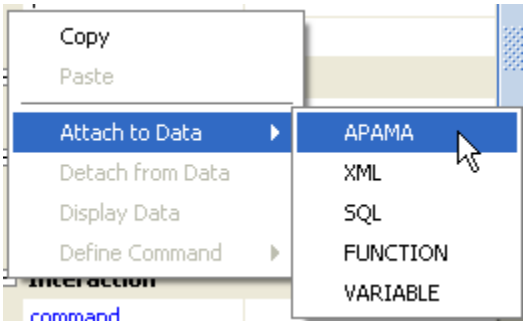
Important: When SQL-based data tables are in use for deployed dashboards, authorization for scenario instances and DataView items does not use scenario authorities (see Administering authorization in Administering dashboard security in *Deploying and Managing Apama Applications*).

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:

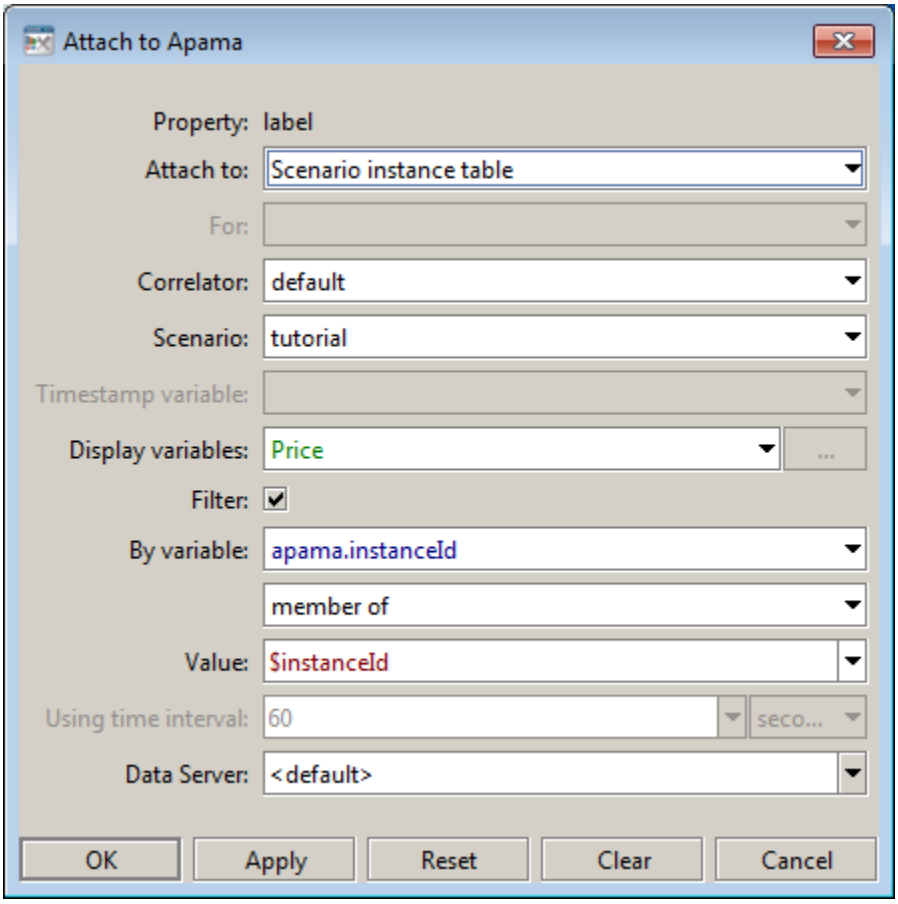
1. Ensure that Builder has been started with the -Y or --`enhancedQuery` command line option.
2. Select the property in the property panel and right click it.

   A popup menu appears.



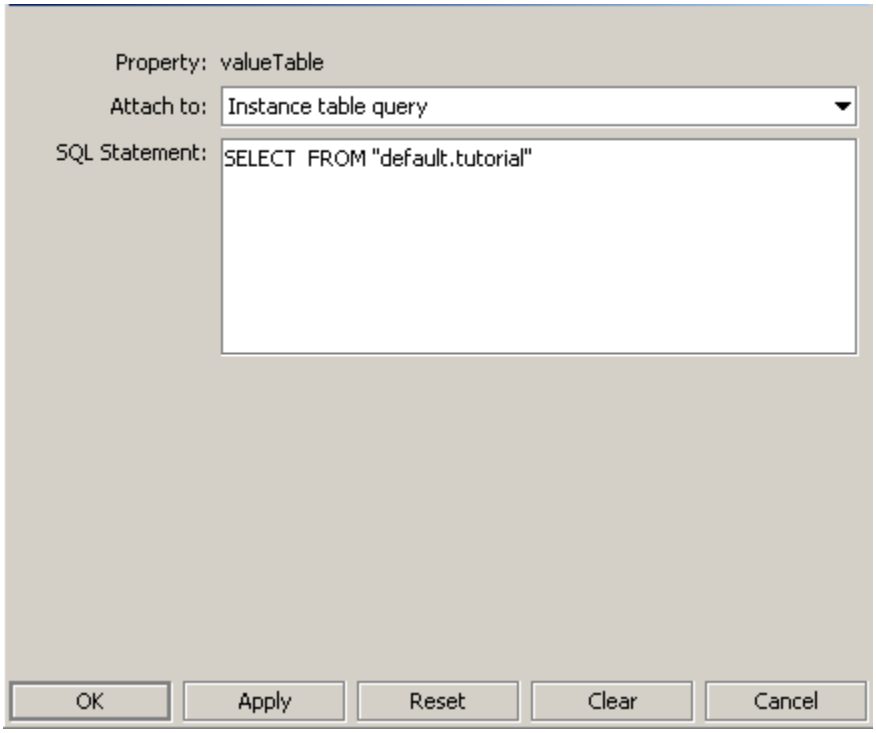
3. In the displayed popup menu pick Attach to Data | Apama.

   This displays the Attach to Apama dialog.



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 varaible 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.

Creating a data attachment

Working with multiple Data Servers

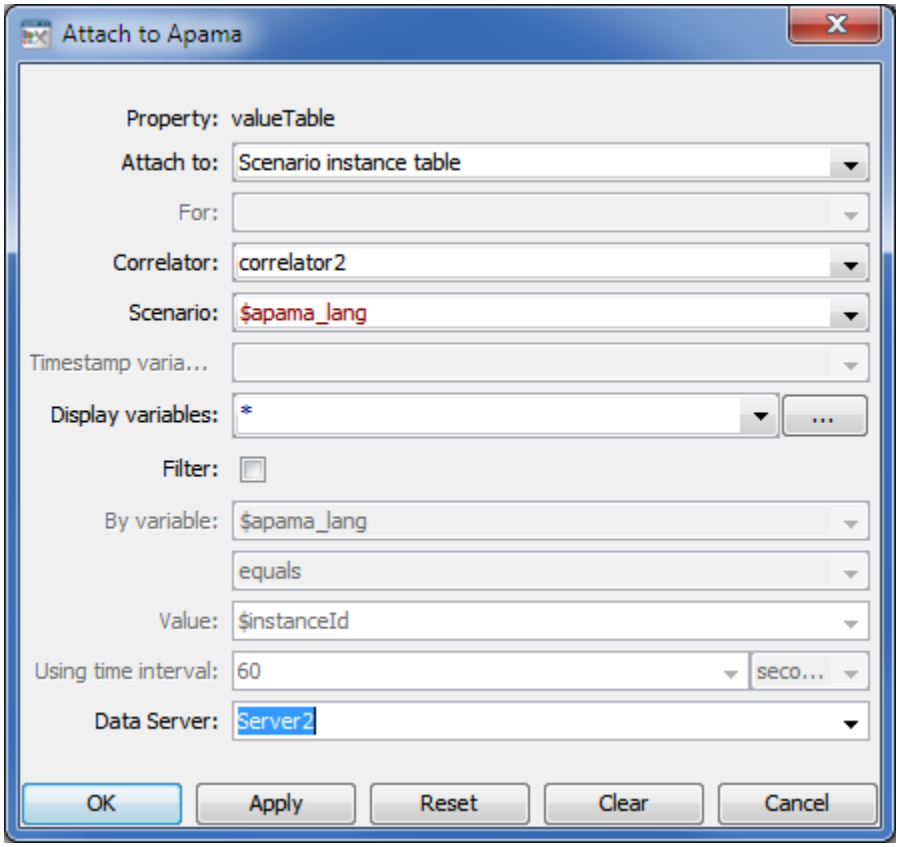
Deployed dashboards have a unique associated default Data Server or Display Server. For Web-based deployments, this default is specified in the Startup and Server section of the Deployment Configuration Editor. For Viewer deployments, it is specified upon Viewer startup. By default, the data-handling 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:

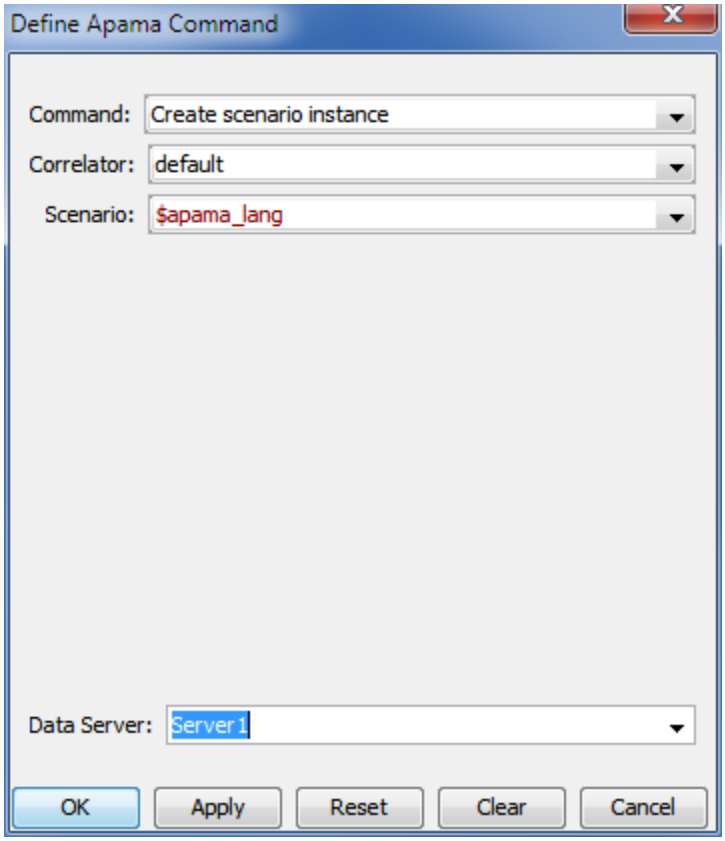
- Builder with multiple Data Servers. See "Builder with multiple Data Servers" on page 72.
- Viewer with multiple Data Servers. See "Viewer with multiple Data Servers" on page 73.
- Display Server (thin client) deployment with multiple Data Servers. See "Display Server deployments with multiple Data Servers" on page 75.
- Applet or WebStart deployment with multiple Data Servers. See "Applet and WebStart deployments with multiple Data Servers" on page 76.

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 ToolsOptions in Builder).

The following attachment specifies Server 2 in the Data Server field at the bottom of the dialog:



The following command specifies Server 1 in the Data Server field at the bottom of the dialog:



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 75.

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:

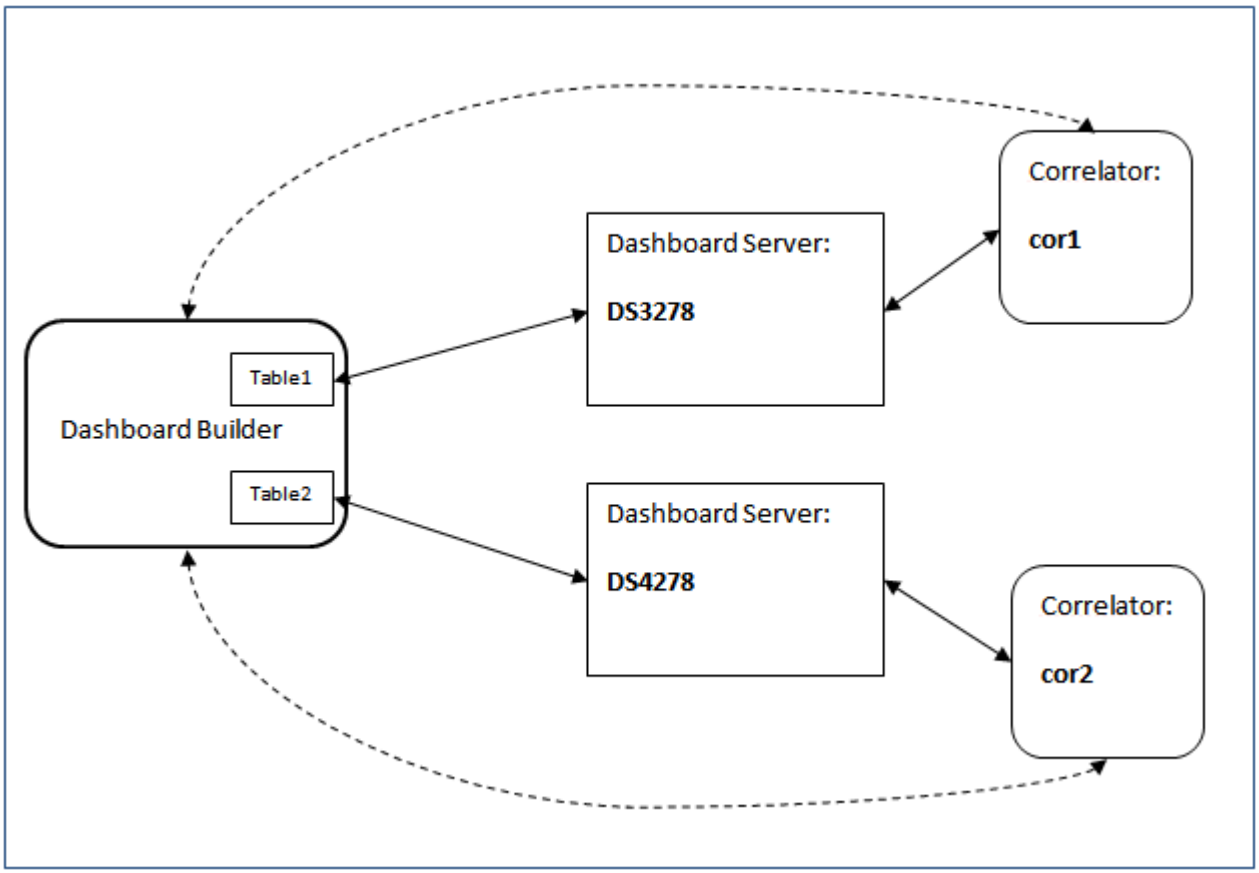
```
--namedServer Server1:ProductionHost_A:3278 --namedServer Server2:ProductionHost_B:4278 --namedServer
Server3:ProductionHost_C:5278
```

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

Creating a data attachment

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:

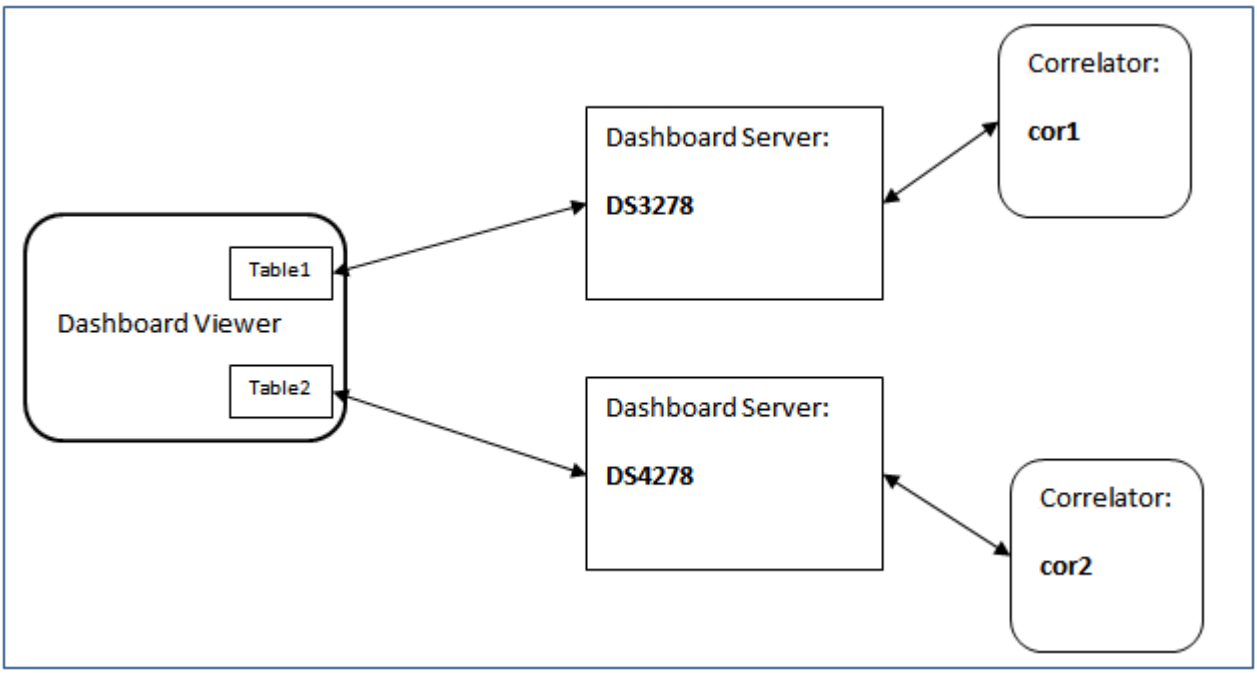
```
--namedServer Server1:ProductionHost_A:3278 --namedServer Server2:ProductionHost_B:4278 --namedServer Server3:ProductionHost_C:5278
```

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

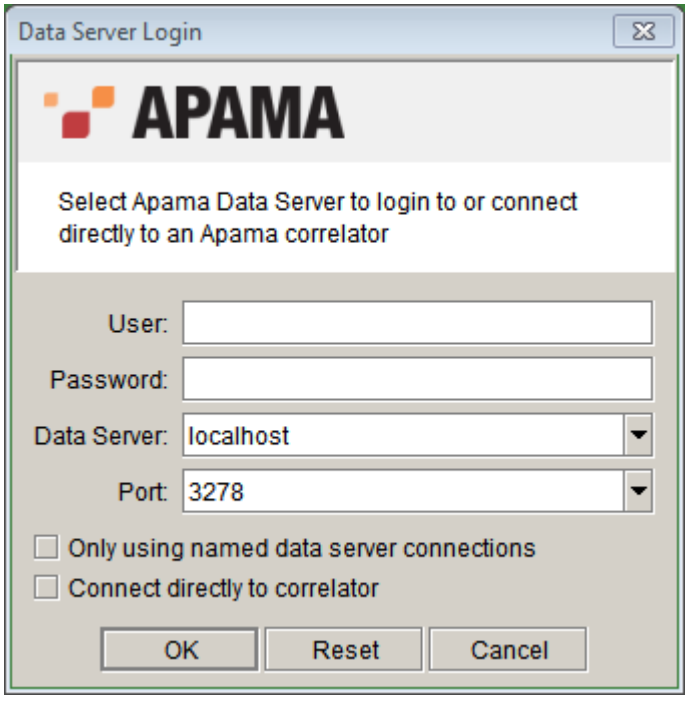
Working with multiple Data Servers

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:

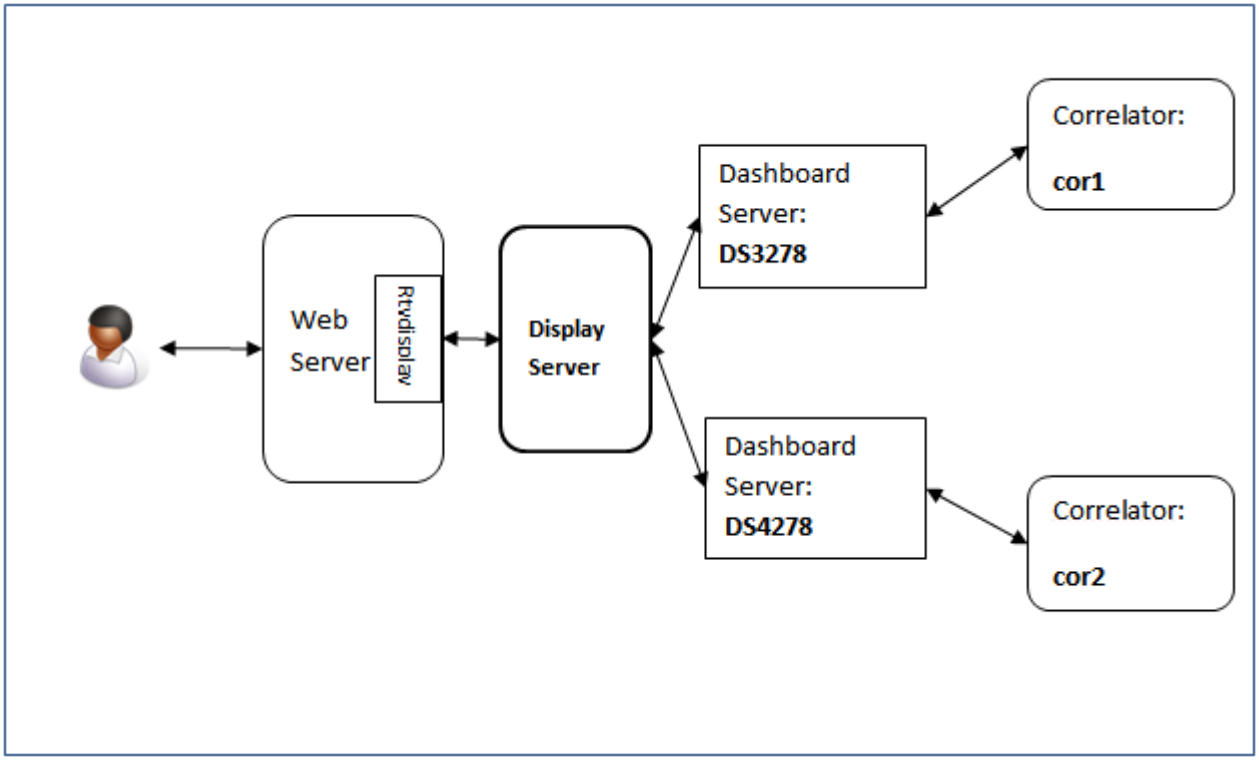
```
--namedServer Server1:ProductionHost_A:3278 --namedServer Server2:ProductionHost_B:4278 --namedServer
Server3:ProductionHost_C:5278
```

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

Working with multiple Data Servers

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.



Note: 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:

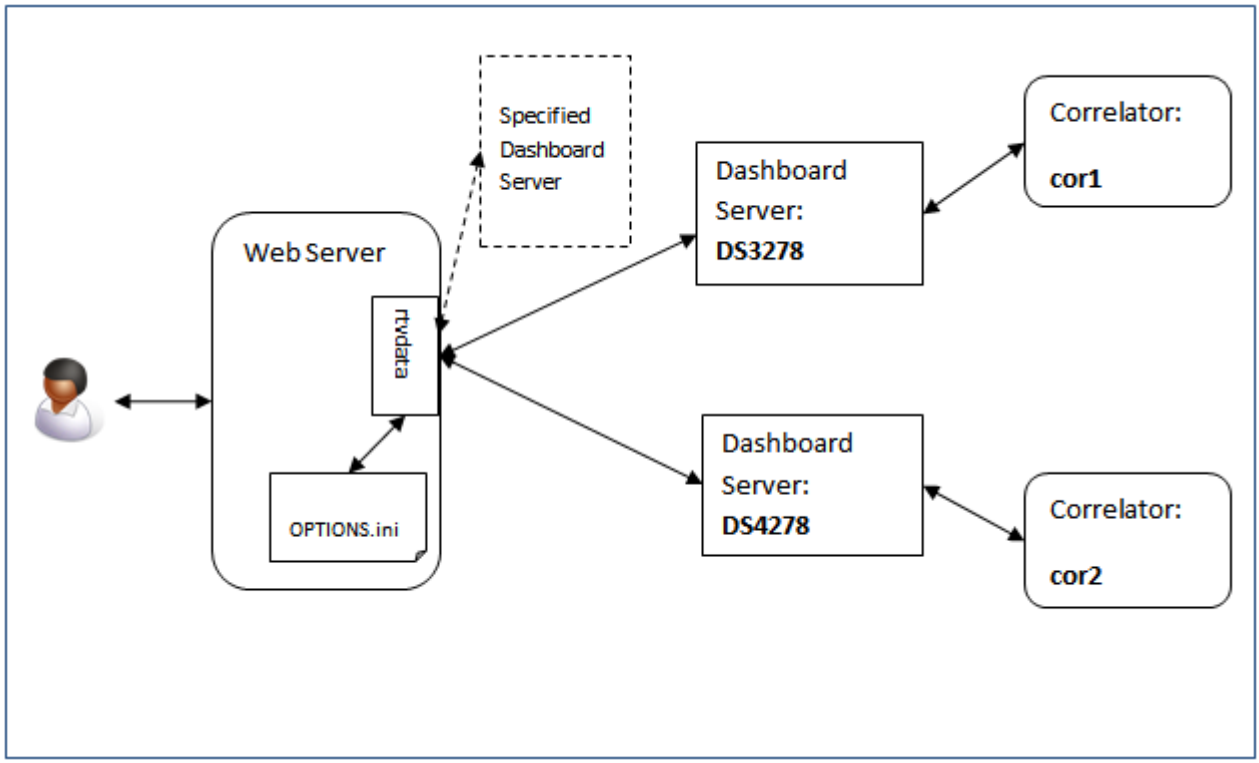
```
--namedServer Server1:ProductionHost_A:3278 --namedServer Server2:ProductionHost_B:4278 --namedServer
Server3:ProductionHost_C:5278
```

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

Working with multiple Data Servers

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.

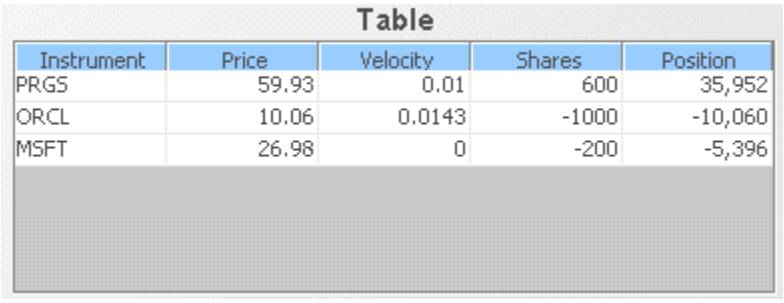
Working with multiple Data Servers

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. You can disable sorting by unchecking the property `showSortIconFlag`.

Common tasks related to tables are covered in the following sections:

"Creating a scenario summary table" on page 78

"Filtering rows of a scenario summary table" on page 80

"Performing drilldowns on tables" on page 81

"Specifying drill-down column substitutions" on page 84

"Hiding table columns" on page 87

"Using pre-set substitution variables for drill down" on page 88

"Formatting table data" on page 88

"Colorizing table rows and cells" on page 89

"Setting column headers" on page 91

"Using rotated tables" on page 92

Detailed reference information on tables is provided in "Table Objects" in the *Apama Dashboard Property Reference*.

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.

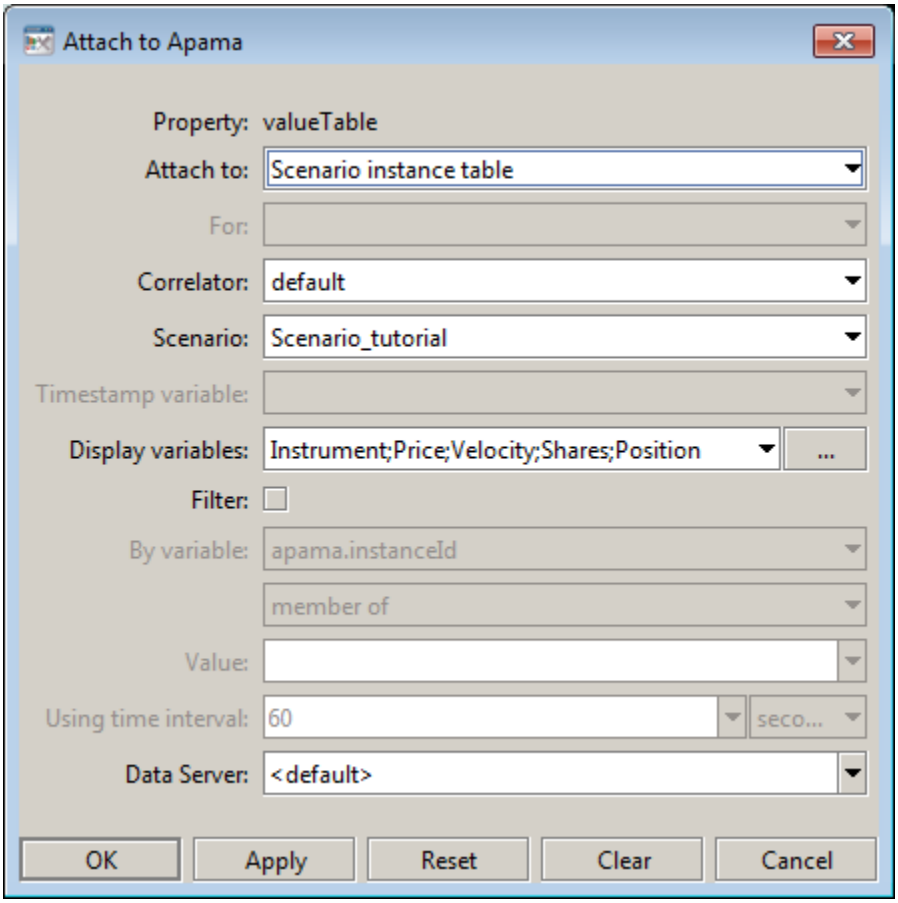
To create a summary table for a scenario, you add a table object to a dashboard and attach its `valueTable` property to a scenario instance table. When you define the attachment, you can select the scenario variables to be displayed; these will be the columns of the table. You can also specify a filter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

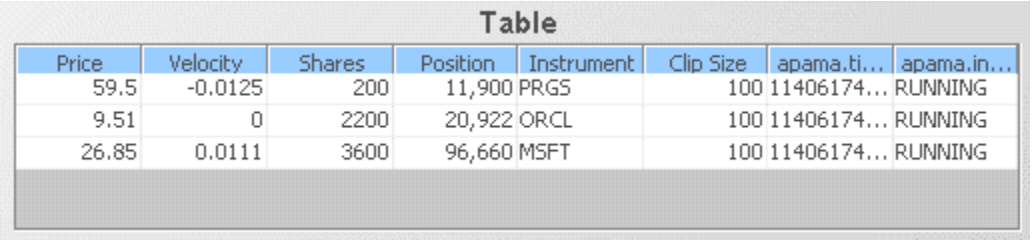
   This displays the Attach to Apama dialog. Attach the table object's `valueTable` property to a scenario instance table, for example as follows:



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.



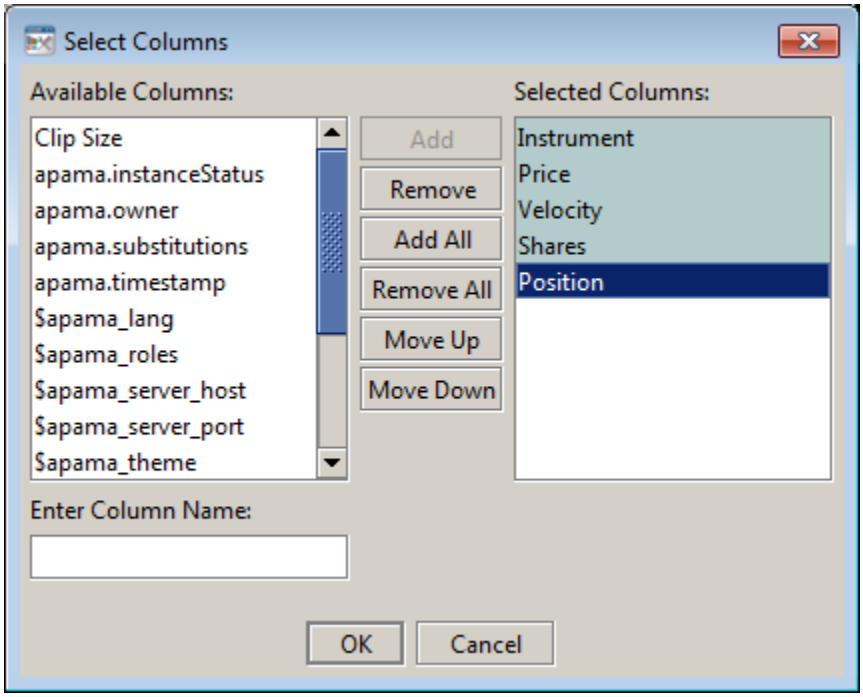
   Often, you will not want to display all scenario variables or the special fields in a summary table. The steps that follow show how to specify the variables to be displayed.

5. Double click on 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.

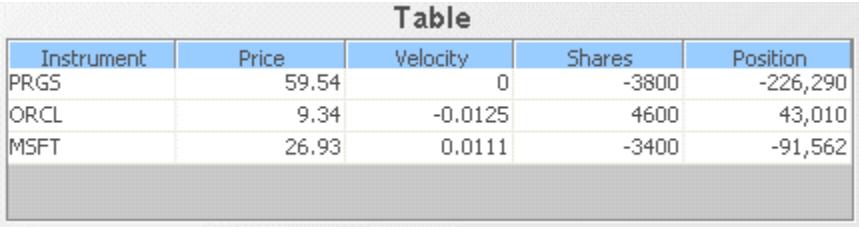
6. Click on the "…" button to display the Select Columns dialog.

7. In the **Select Columns** dialog select and order the columns as follows.



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.



Table

Instrument	Price	Velocity	Shares	Position
PRGS	59.54	0	-3800	-226,290
ORCL	9.34	-0.0125	4600	43,010
MSFT	26.93	0.0111	-3400	-91,562

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`.

Using table objects

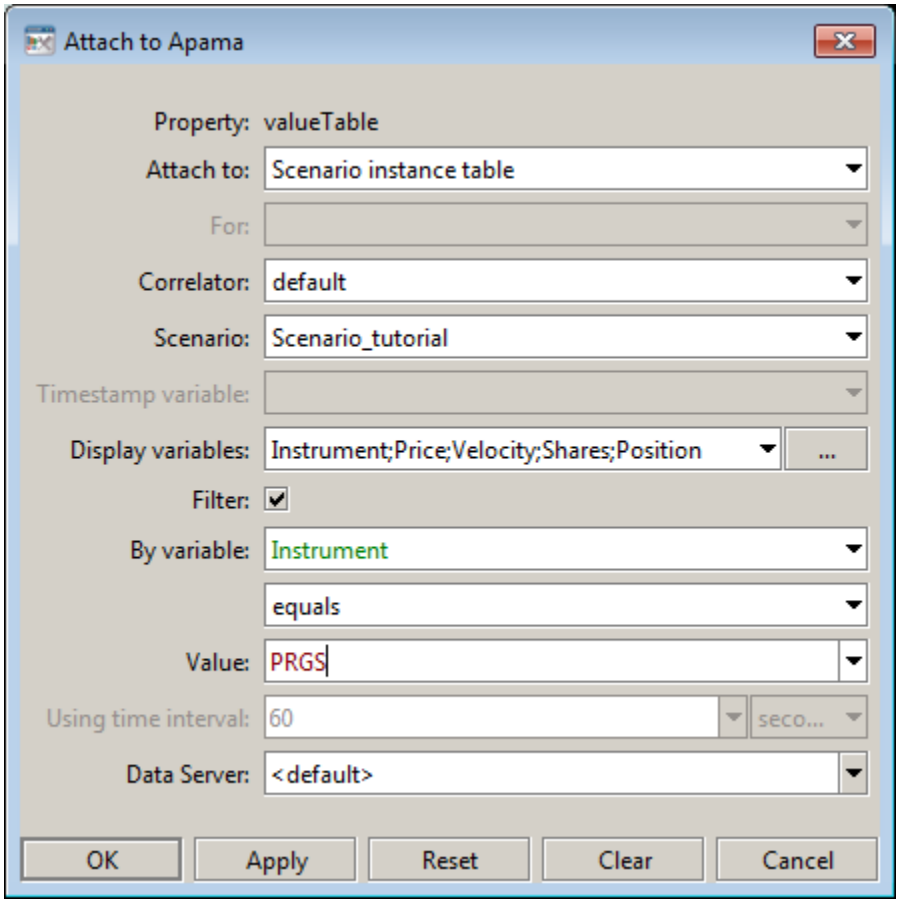
Filtering rows of a scenario summary table

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.

Follow these steps 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:



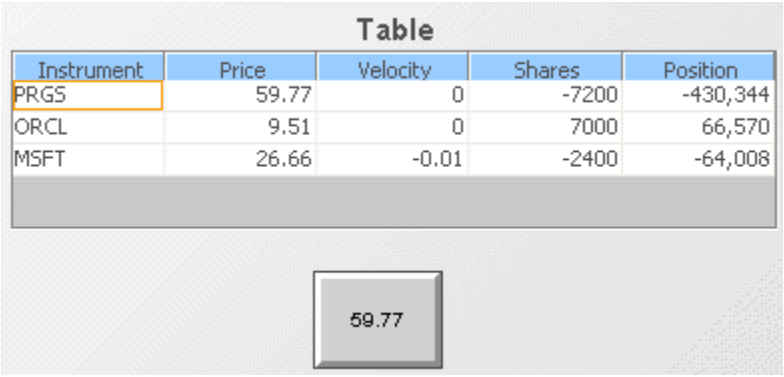
This example filters the table's contents to display only the instance for which the value of the scenario variable `Instrument` equals `APMA`.

Using table objects

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 on 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.

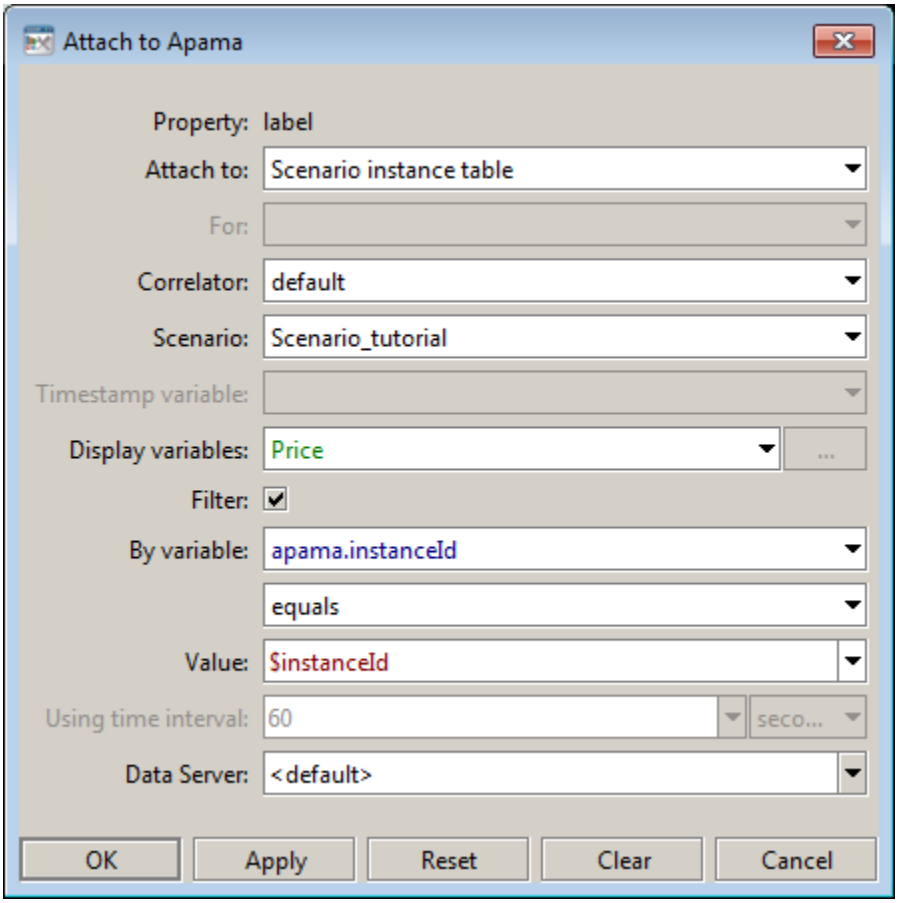
To specify a drilldown as in the example above, perform these steps:

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 on 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 on 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 the appendix Drilldown Specification in the *Dashboard Property Reference*.

   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 84 describes how to override this default setting.

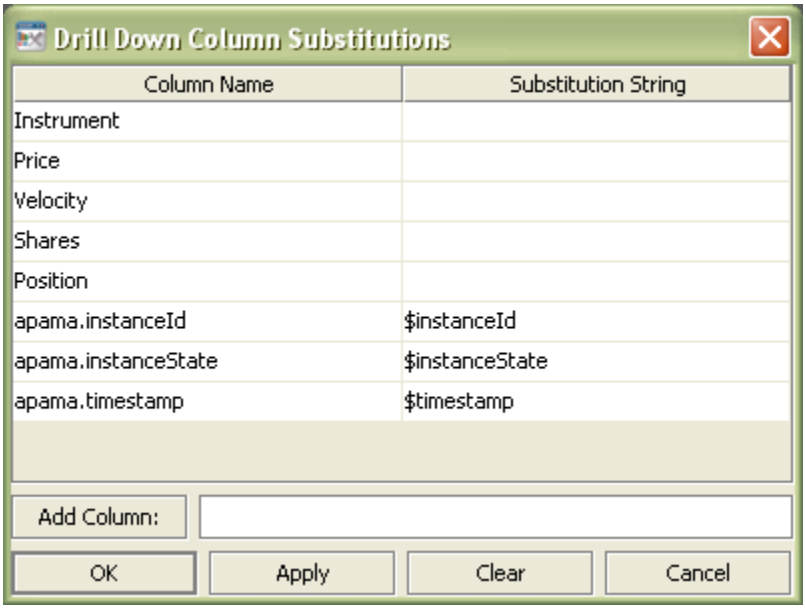
Using table objects

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.

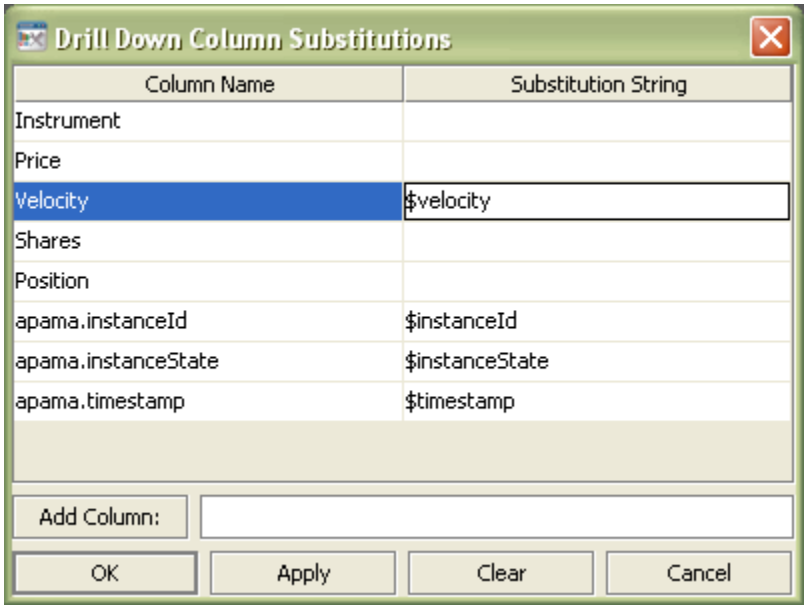
1. Select the table object and double click on the `drillDownColumnSubs` property.

   The Drill Down Column Substitutions dialog displays.



   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.

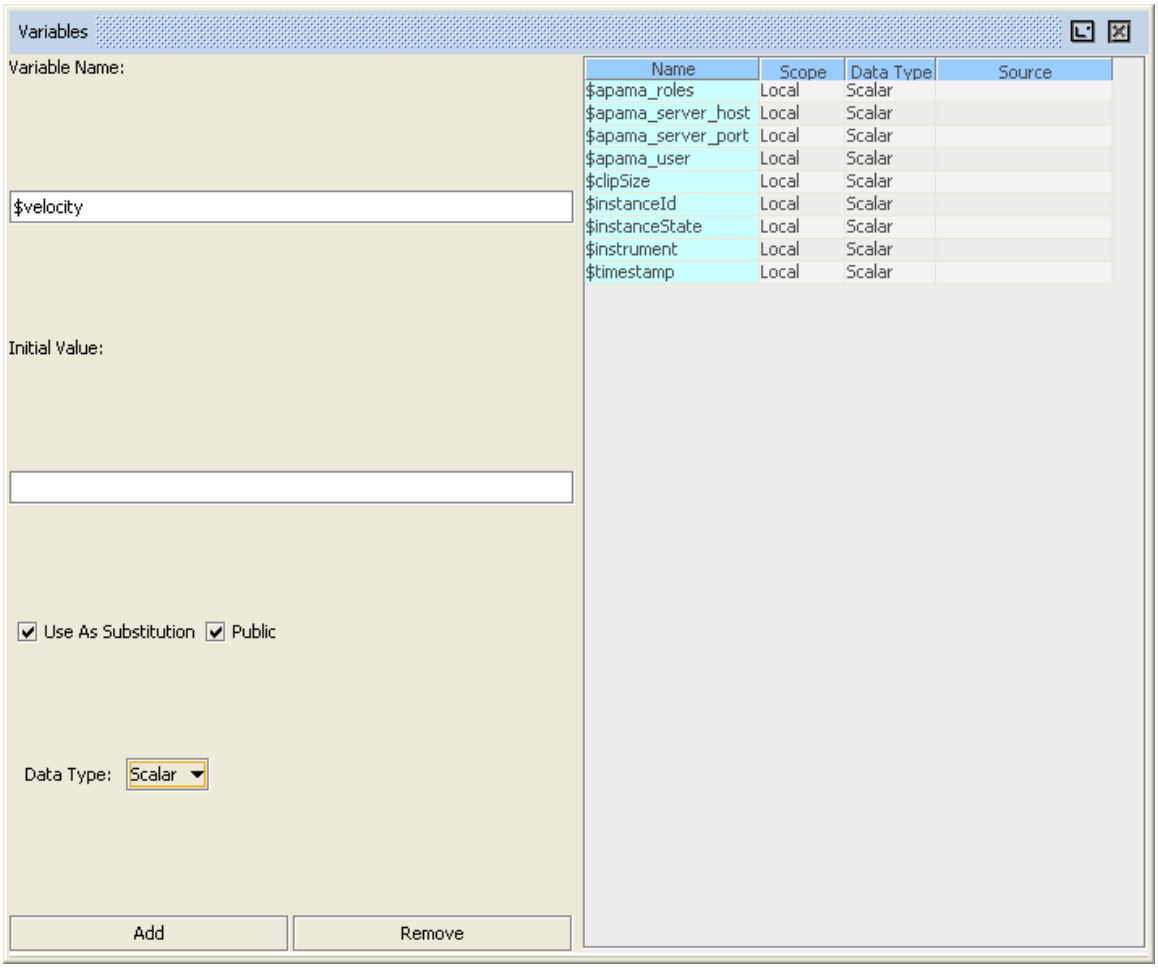


3. Click the OK button to close the dialog.

   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.

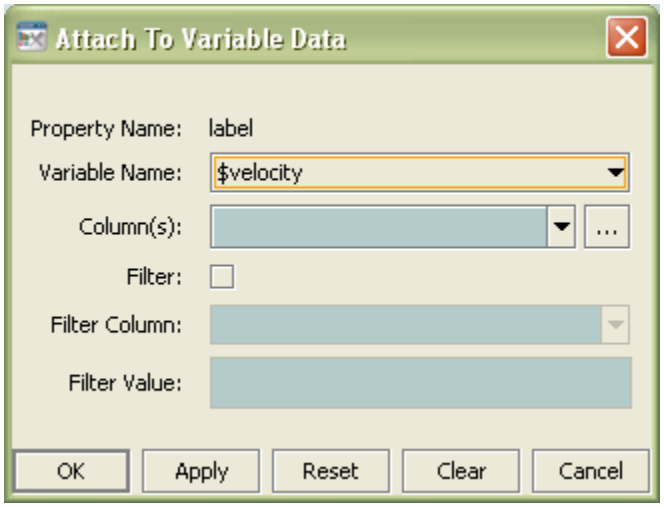
   Add the substitution variable `$velocity`.



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 on 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 `$velocity`. When you double click on a row in the table, the dashboard performs the drilldown and sets `$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.

Using table objects

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 table objects

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.

Using table objects

Formatting table data

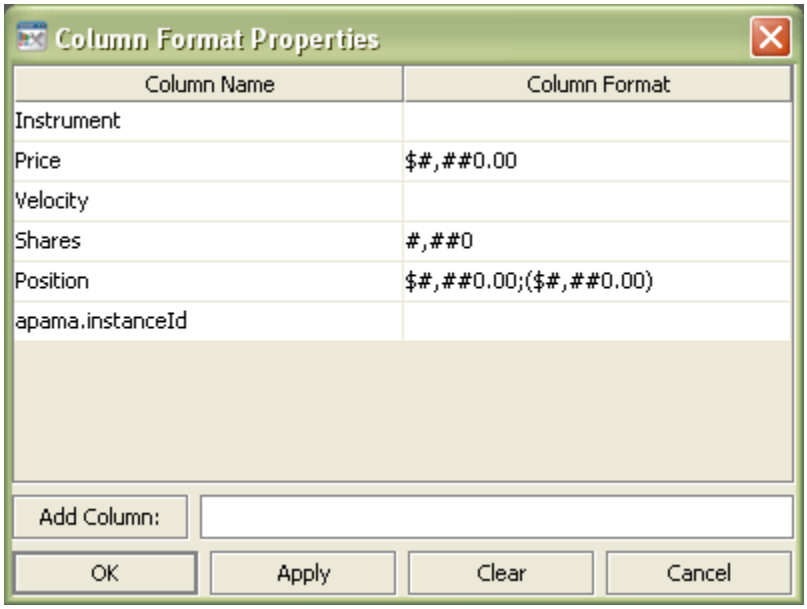
The table object allows formatting attributes to be specified for each column in a table. Double-click on Formatted Table in the tutorial main page to see the following table:

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:



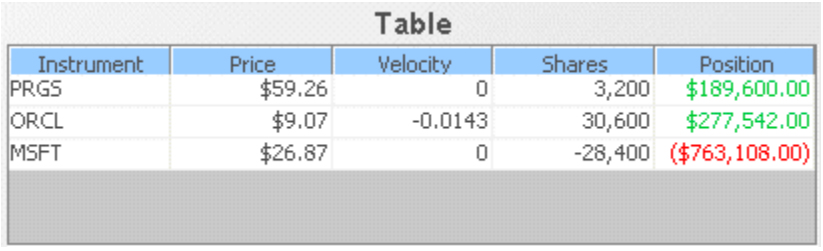
To format a column, select the column in the Column Name field and either select, or type, a format string in the Column Format field.

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.

Using table objects

Colorizing table rows and cells

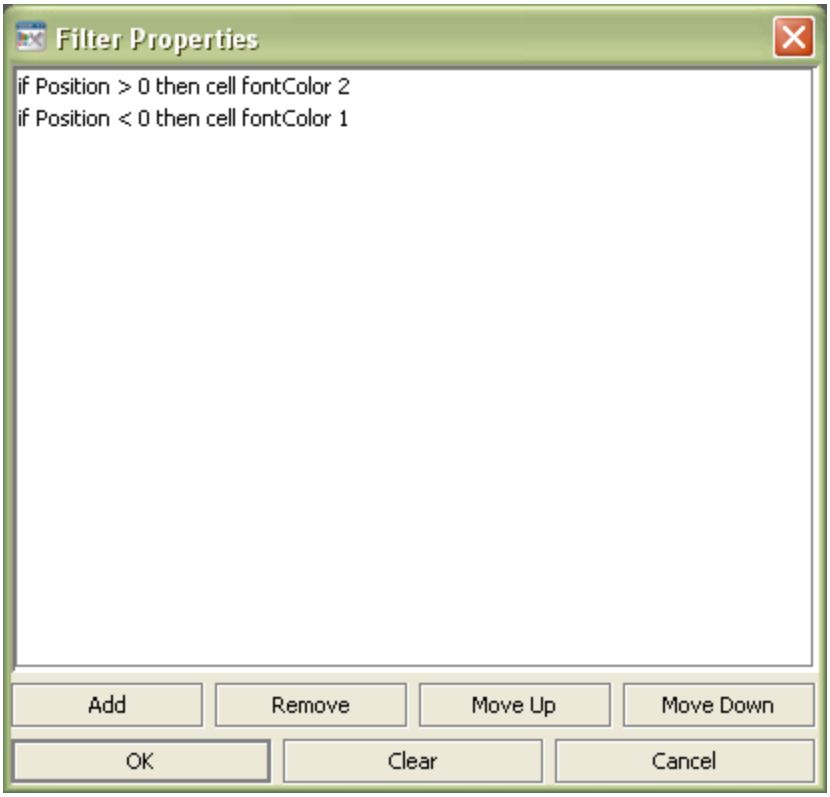
The table object allows the color attributes of rows and cells to be set based on the value of a scenario variable or DataView field. Double-click Colored Table on the tutorial main page to see the following table, which shows a typical use for setting color attributes.



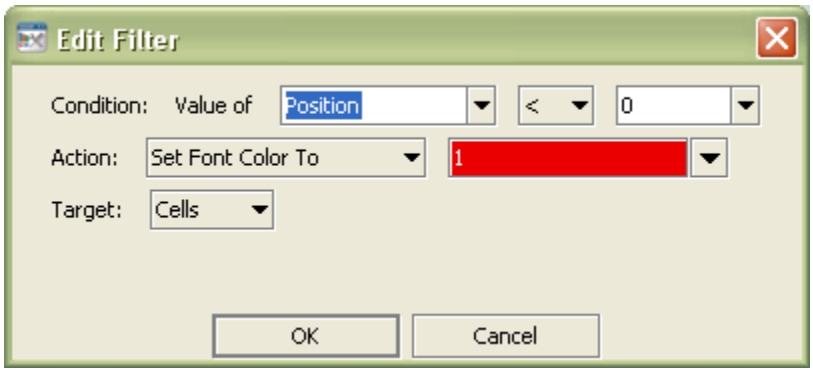
Here the Position cell is shown with green text if the position is positive and red text if it is negative. Colorizing a table can make it much easier to identify values of interest. Colorizing attributes are specified by setting the `filterProperties` property.:

1. Double click on the `filterProperties` property.

   The **Filter Properties** dialog displays.



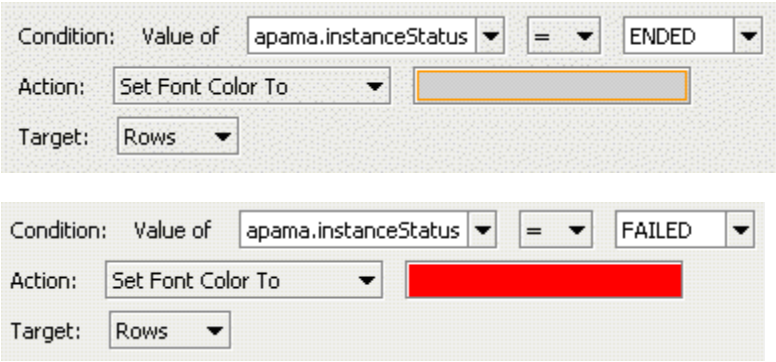
2. Double click on a filter to edit it or click on the Add button to add a new filter.



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.



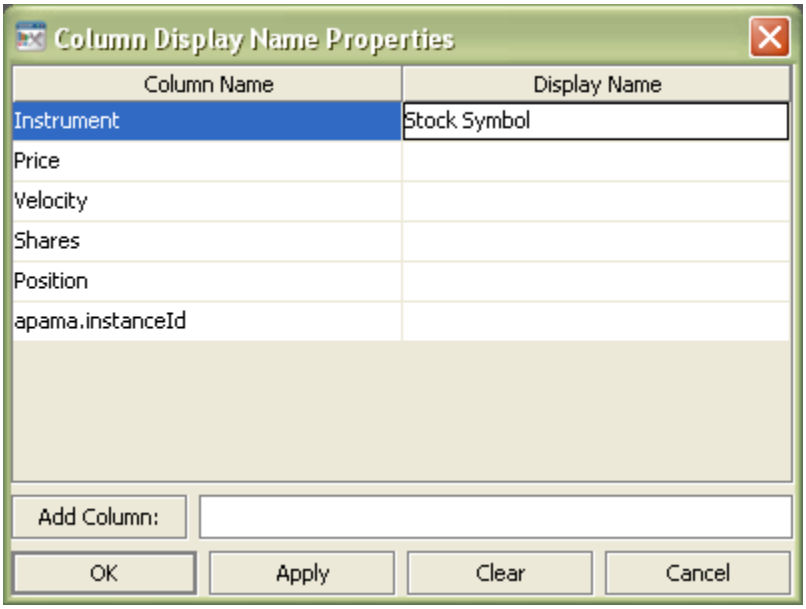
Using table objects

Setting column headers

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



   In this example, 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 a \n in the display name such as "`Stock\nSymbol`".

Stock Symbol	Price
PRGS	60.35
ORCL	10.37
MSFT	27.32

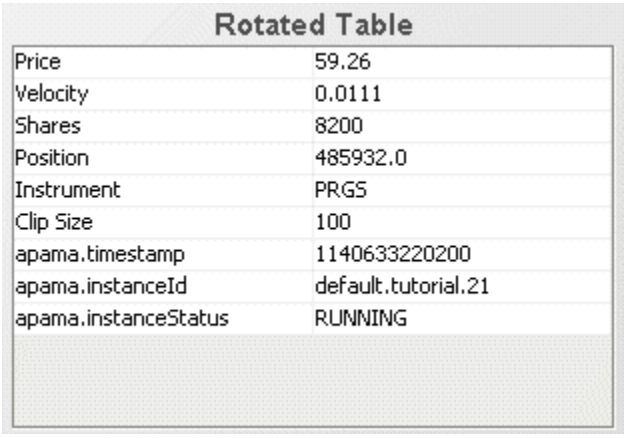
Using table objects

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, perform the following steps:

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 78).



Rotated Table

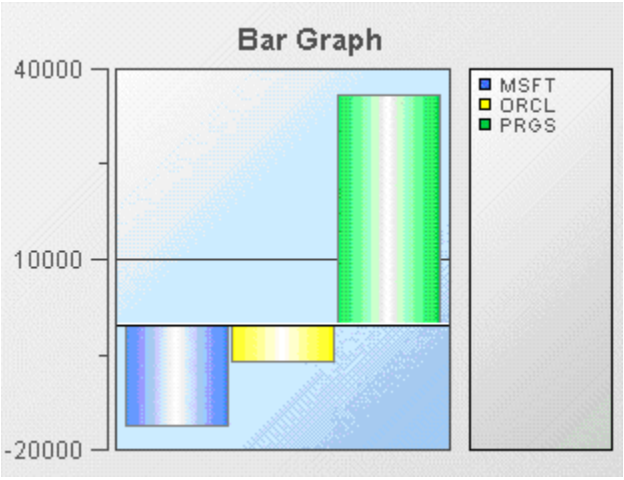
Price	59.26
Velocity	0.0111
Shares	8200
Position	485932.0
Instrument	PRGS
Clip Size	100
apama.timestamp	1140633220200
apama.instanceId	default.tutorial.21
apama.instanceStatus	RUNNING

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.

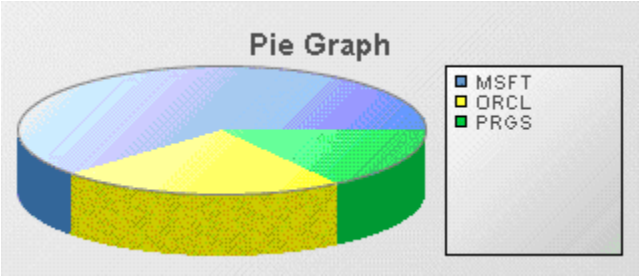
Using table objects

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 drilldowns similar to those supported by table objects.

Common tasks related to pie and bar charts are covered in the following sections:

"Creating a summary pie or bar chart" on page 93

"Using series and non-series bar charts" on page 94

"Performing drilldowns on pie and bar charts" on page 95

Detailed reference information on graphs, including pie and bar charts, is provided in "Graph Objects" in the *Apama Dashboard Property Reference*.

Attaching Dashboards to Correlator Data

Creating a summary pie or bar chart

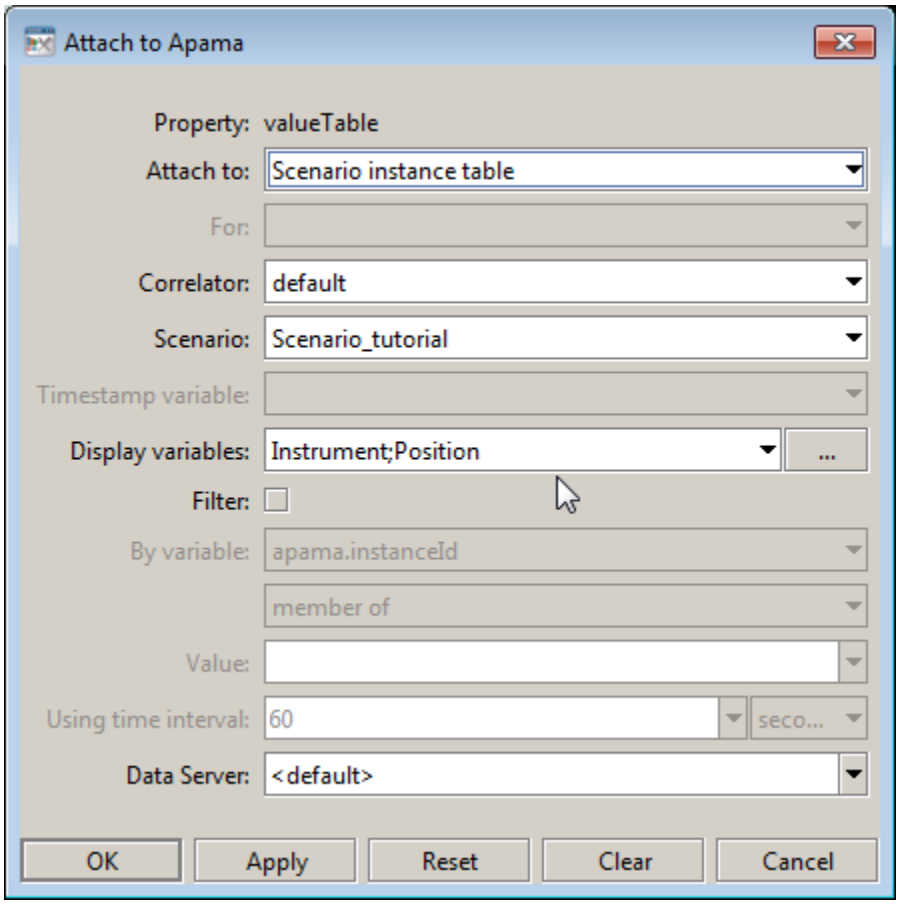
To create a summary pie or bar chart for a scenario or DataView, you add an instance of the object to a dashboard and attach its `valueTable` property to a DataView or scenario instance table. When you define the attachment, you can select the variable to be charted as well as the label to be used for 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 as detailed in the following section.

Using pie and bar charts

Using series and non-series bar charts

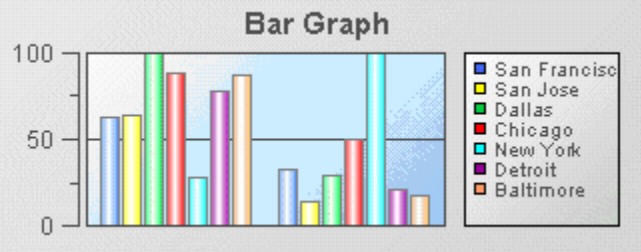
Data in bar charts can be displayed as both series and non-series data. This is determined by the `rowSeriesFlag` property.

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.

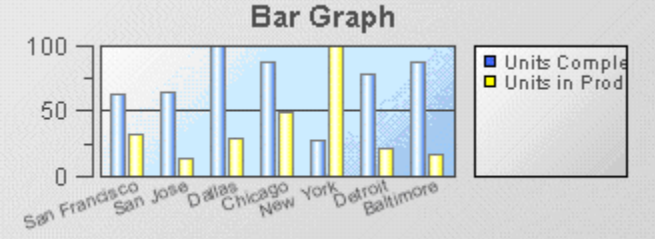
1. Create a new dashboard, select the Graphs tab in the Object Palette, and add a Bar Graph object to the dashboard canvas.

   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.



Using pie and bar charts

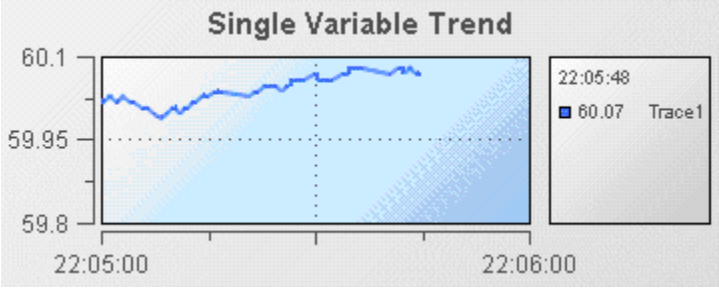
Performing drilldowns on pie and bar charts

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 pie and bar charts

Using trend charts

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:



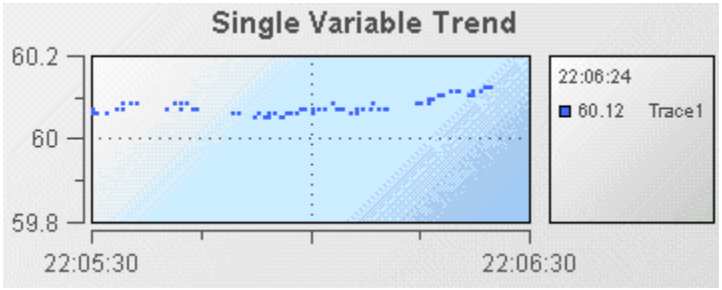
In this sample, a single trend line is displayed to show the value of the `Price` variable of an instance of the tutorial scenario.

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.

Common tasks related to trend charts are covered in the following sections:

"Creating a scenario trend chart" on page 97

"Charting multiple variables" on page 100

"Adding thresholds" on page 105

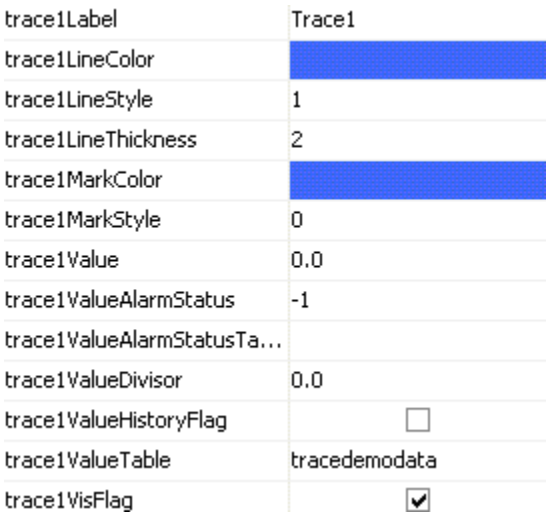
"Configuring trend-data caching" on page 108

Detailed reference information on trend charts is provided in "Trend Objects" in the *Apama Dashboard Property Reference*.

Attaching Dashboards to Correlator Data

Creating a scenario trend chart

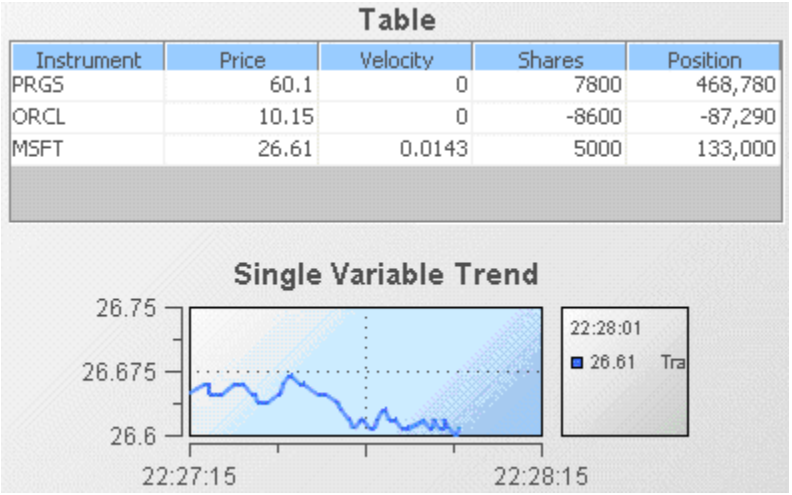
To create a trend chart for a scenario, you add it to a dashboard and set its `traceCount` property to the number of trace lines you want to display. This will cause a set of properties to be added to the property panel for each trace; `trace1` through `trace`*N*. Following are the properties for `trace1`.



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).

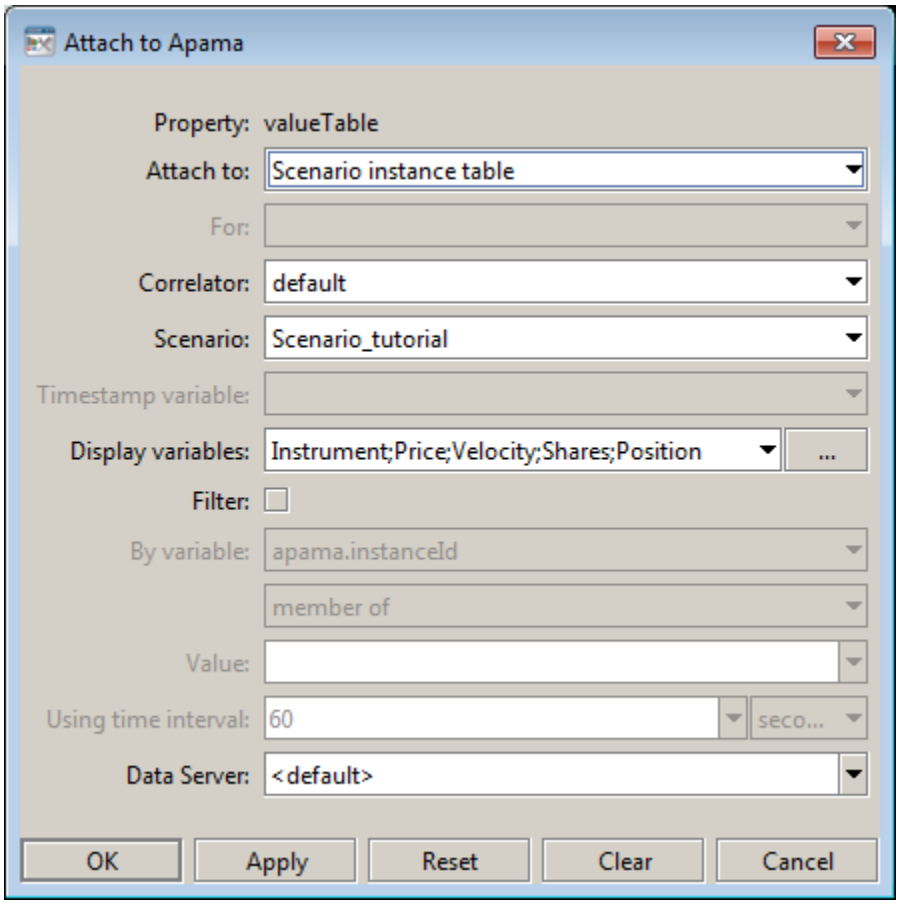
When attaching a trace to a scenario variable, you must specify a filter that identifies the scenario instance the trace will show data for. The filter to identify the instance will typically match on `$instanceId` although other filters can also be used.

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.

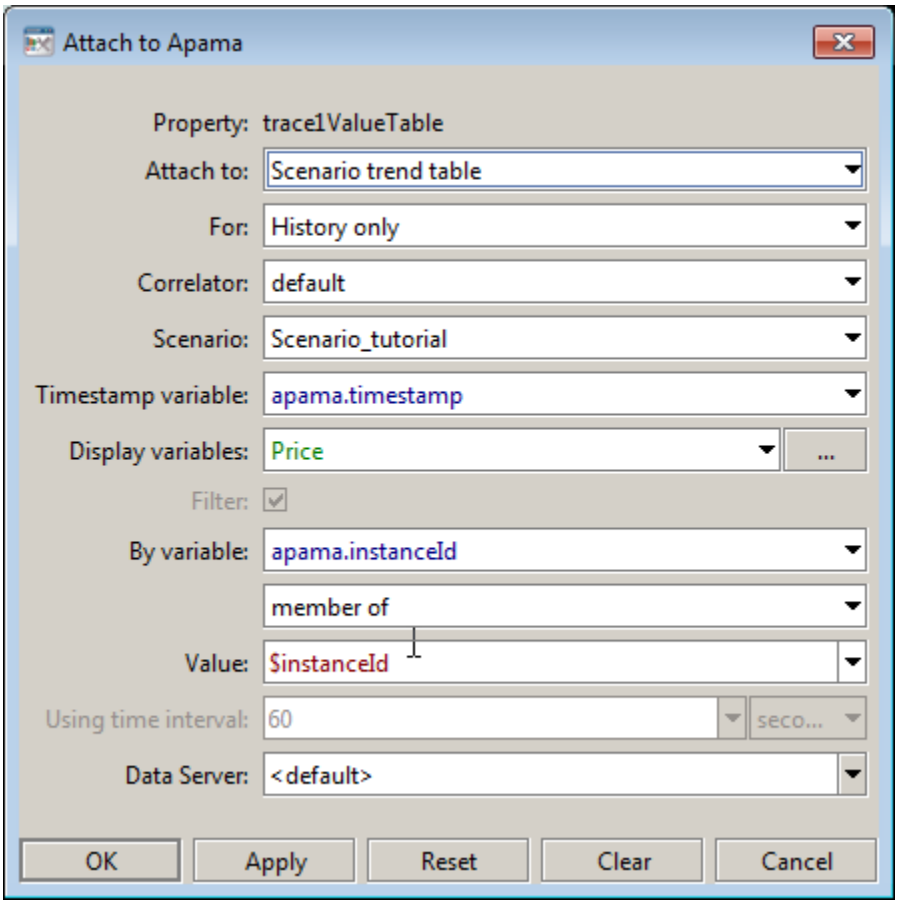
1. From the Tables tab in the Object Palette, select the Table object and add it to the dashboard canvas.
2. With the table object selected, in the Object Properties panel, double click the `valueTable` property and attach it to Apama by specifying the information shown below in the Scenario and Display 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.

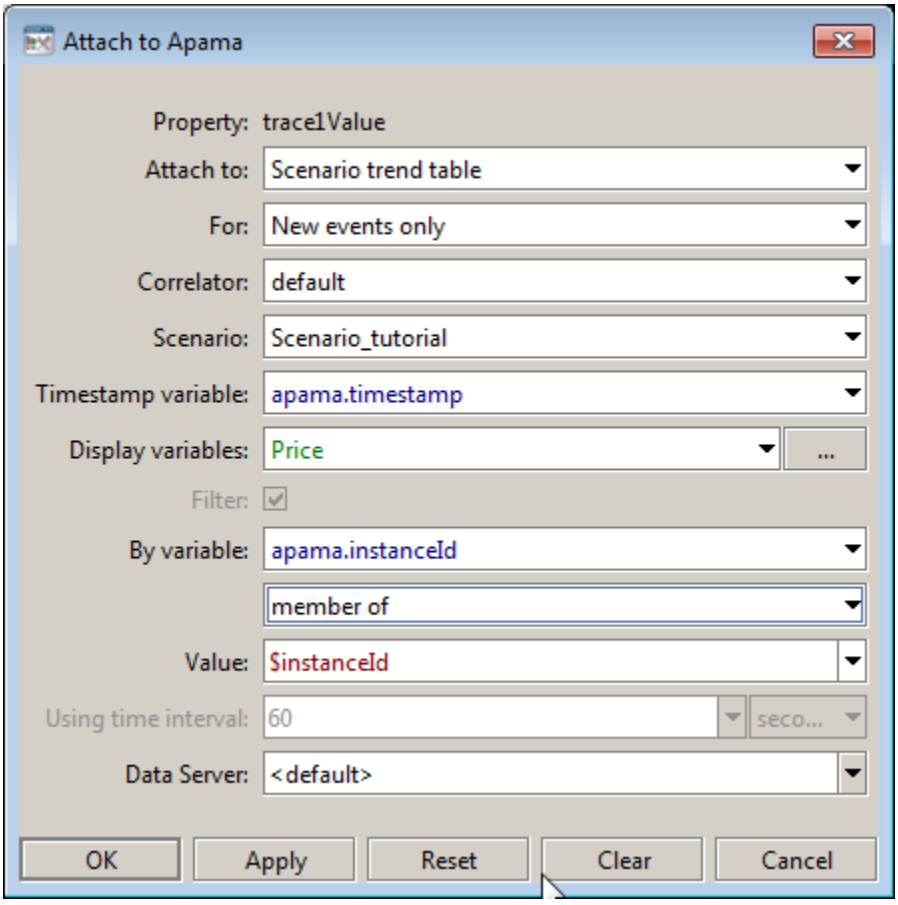
4. With the trend object selected, in the Object Properties panel, double click the trend chart object's `trace1ValueTable` property and attach it to the trend table for the tutorial scenario by specifying values in the fields as shown below.

   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`.



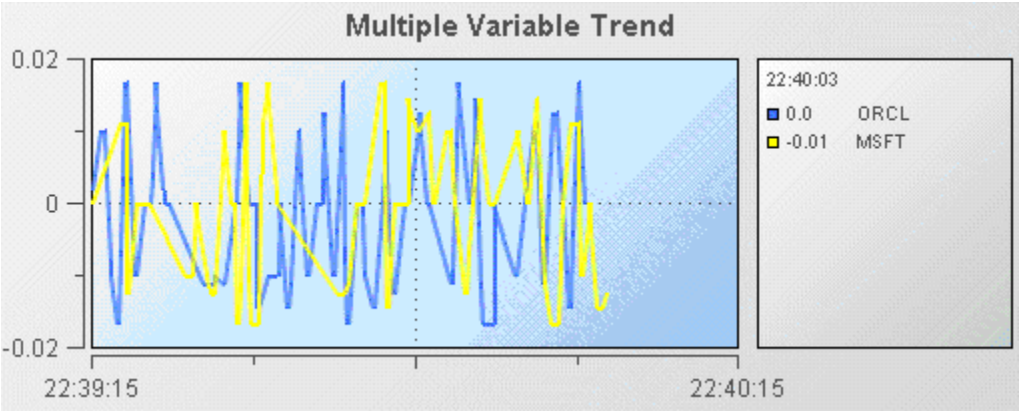
6. Select the trend object's `scrollbarMode` property and change its value to `As Needed`. This will add a scrollbar to the chart allowing you to scroll back in time to view earlier values.
7. Select a scenario instance in the table by double clicking on it. The chart will now begin charting the `Price` 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.

Using trend charts

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.



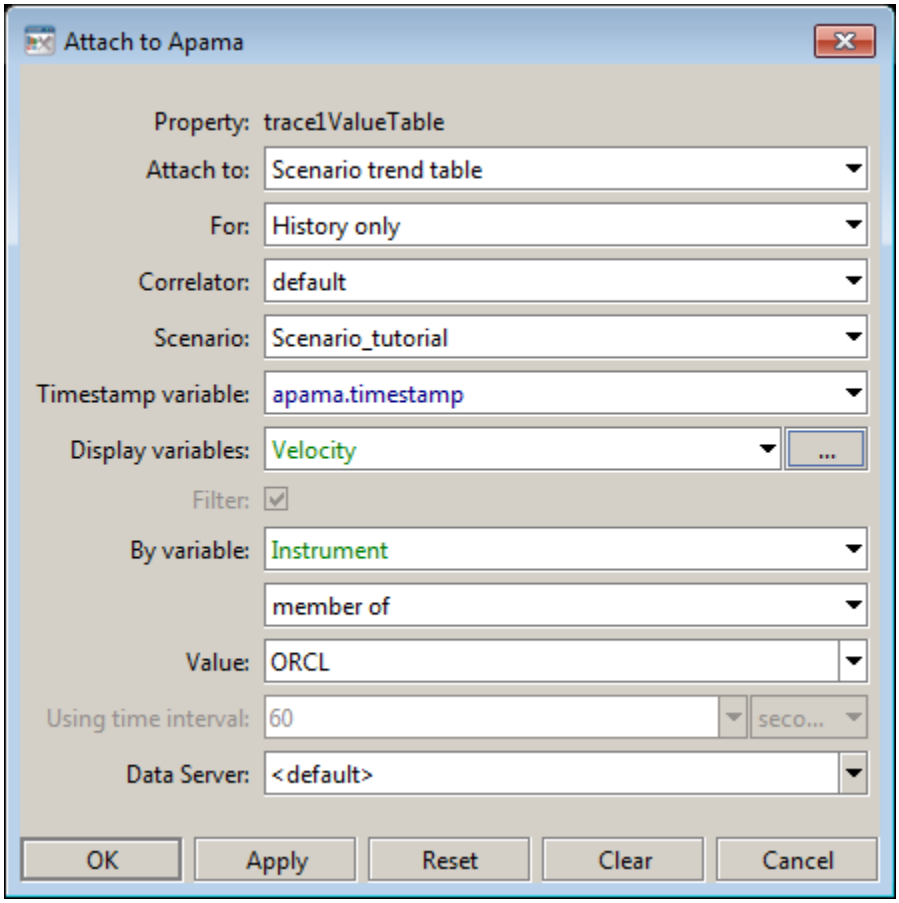
Here the trend chart displays the `Velocity` of the stock price of two instances of the tutorial scenario; one where the Instrument is ORCL and the second where the Instrument is MSFT.

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

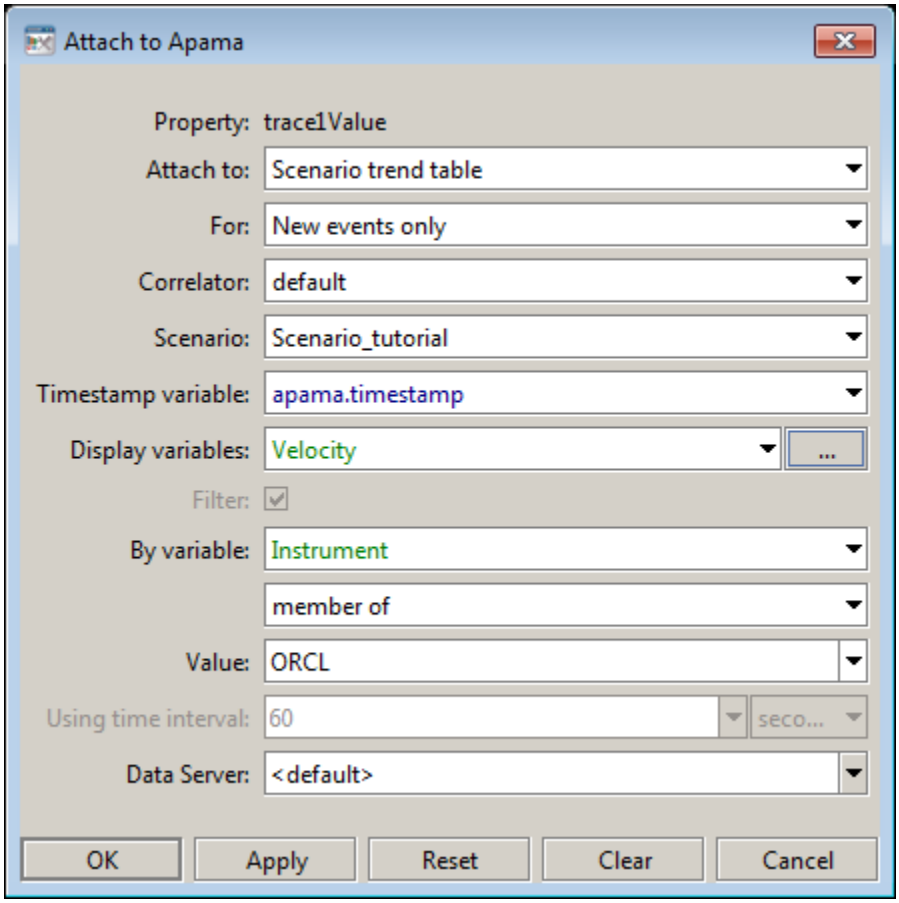
1. From the Trends tab in the object palette, select the Multiple Variable Trend object and add it to the dashboard canvas.

   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.

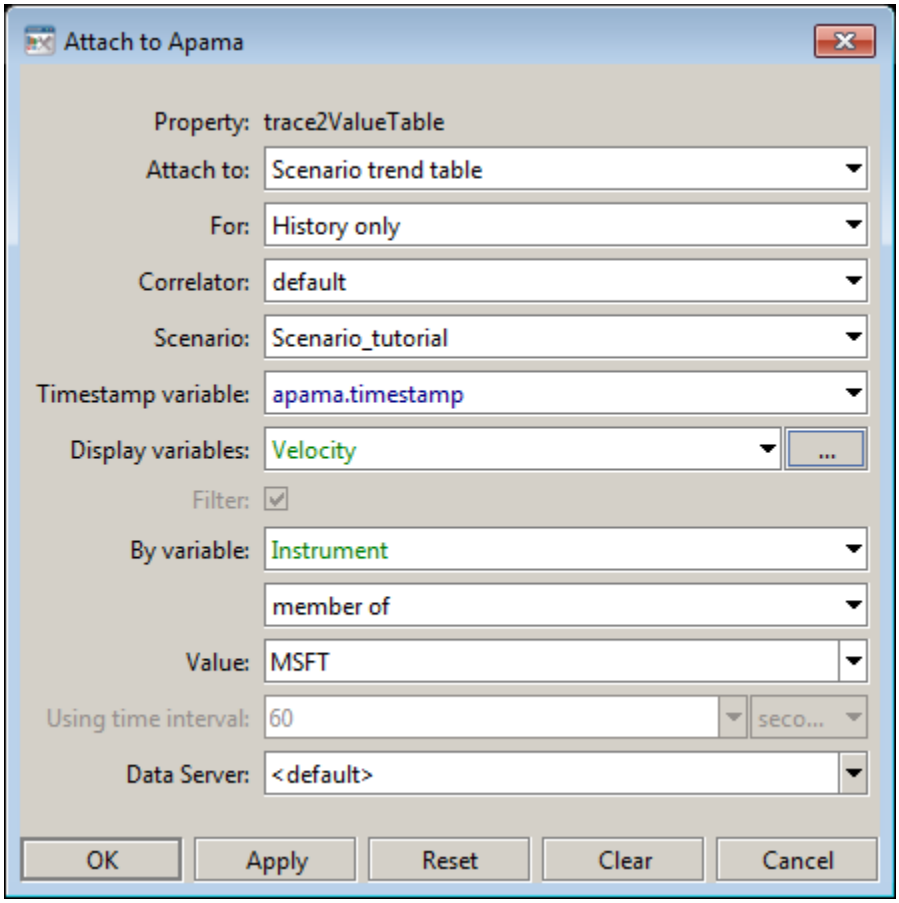
2. With the trend object selected, in the Object Properties panel, double click the trend object's `trace1ValueTable` property and attach it to Apama by specifying the following information:



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:

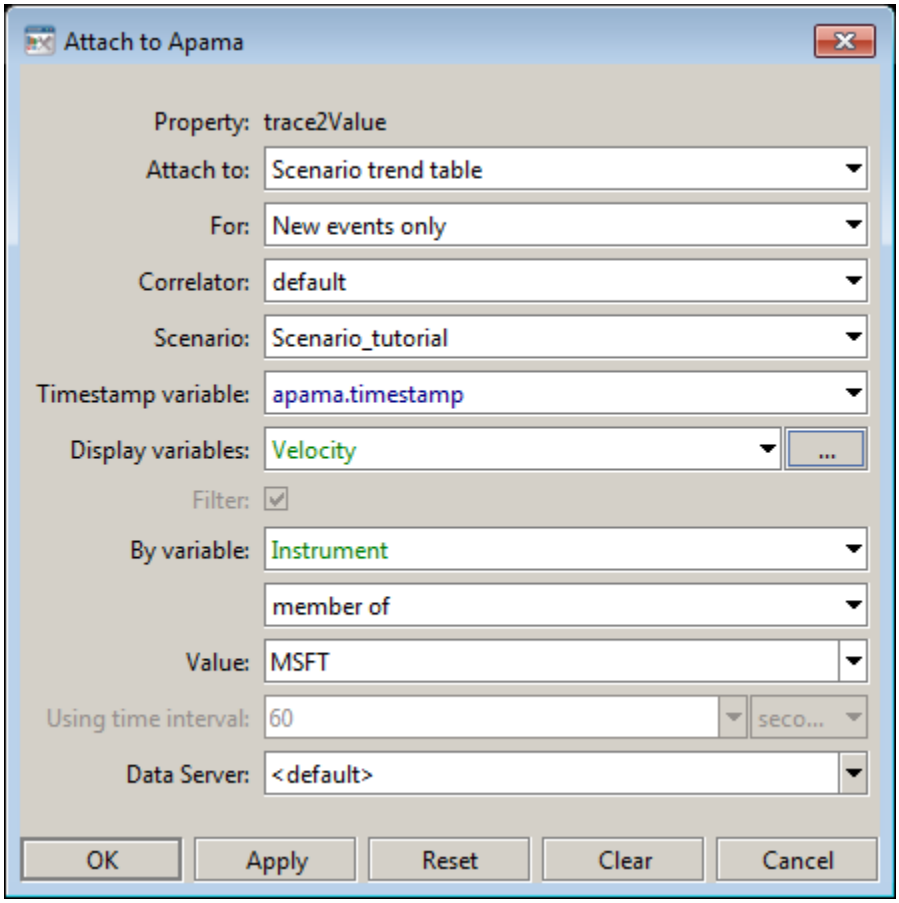


4. Attach the `trace2ValueTable` property to Apama by specifying the following information:



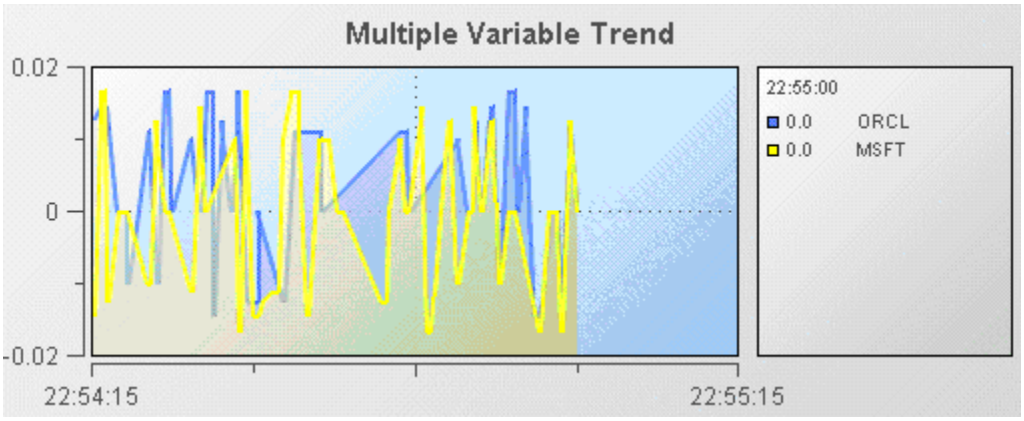
The trend chart will now chart the `Velocity` variable of the instances of the tutorial scenario which match the filters where `Instrument` equals `ORCL` and `MSFT`.

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`.

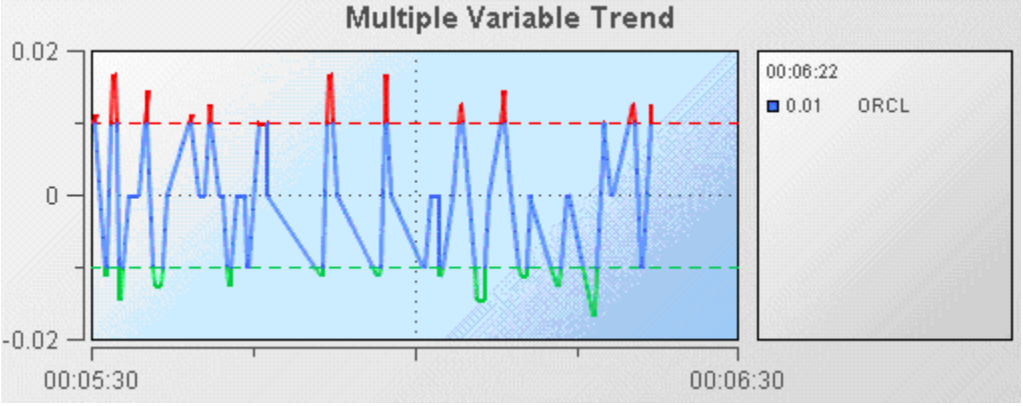
When displaying multiple traces, it is often useful to display them as filled regions. To specify this, select the `traceFillStyle` property and change its value to `Transparent Gradient`. The following illustration shows an example of a trend chart with filled regions.



Using trend charts

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.



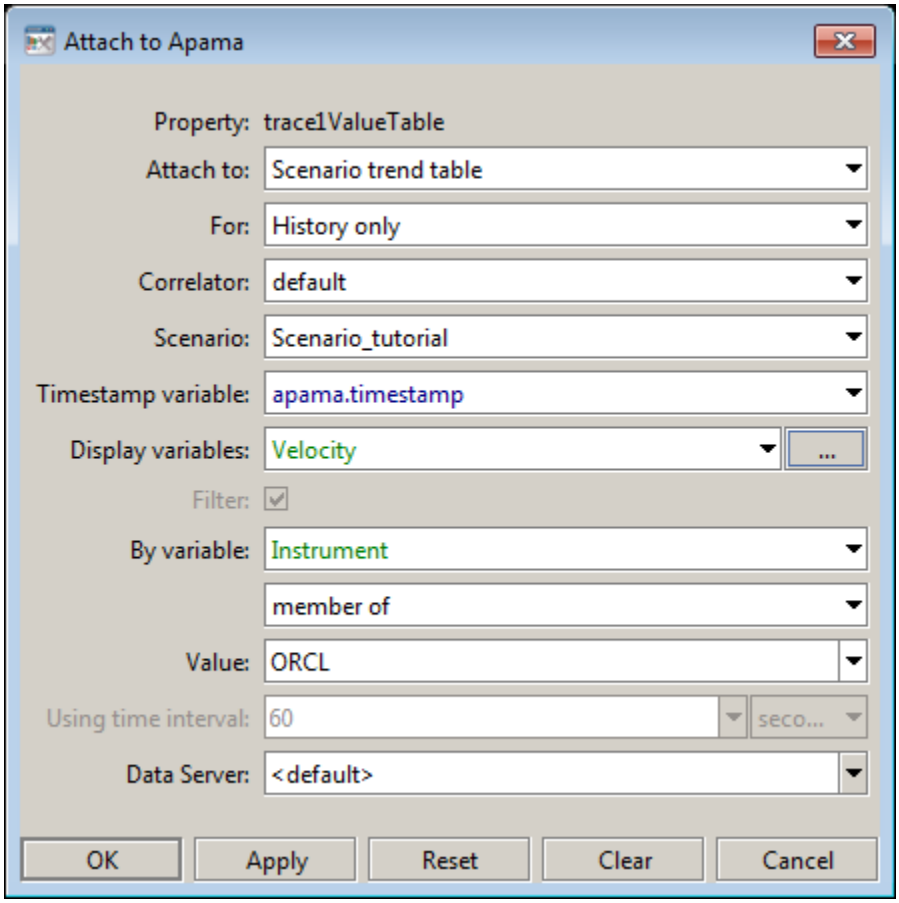
Here the `Velocity` of an instance of the tutorial is being charted and high and low thresholds of .01 and -.01 are being displayed.

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.

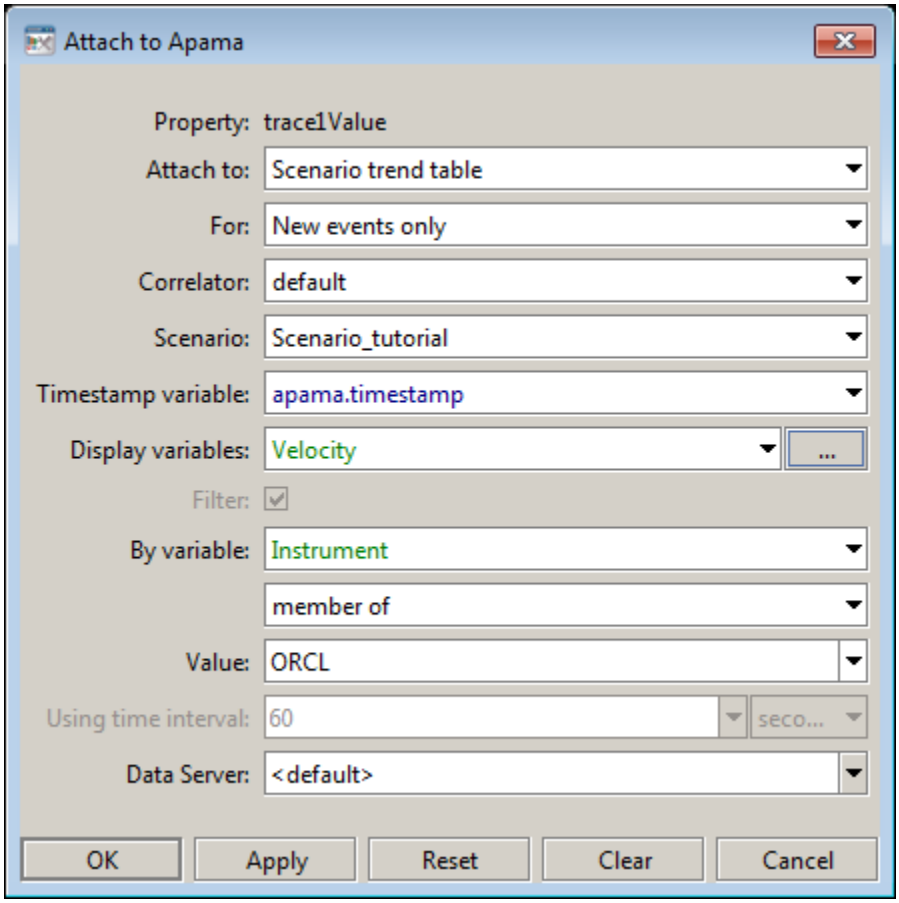
valueHighAlarm	0.01
valueHighAlarmEnabledFlag	☑
valueHighAlarmLineVisFlag	☑
valueHighAlarmMarkColor	
valueHighAlarmMarkStyle	0
valueHighAlarmTraceColor	
valueHighAlarmTraceStyle	1

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.

Using trend charts

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.

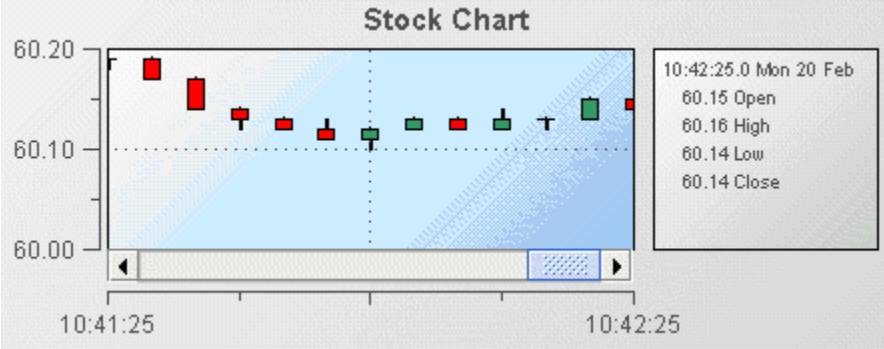
Advanced users can override the default caching behavior on a given server, and control caching in order to reduce memory consumption on that server, or in order to cache variables that are not cached by default, such as non-numeric variables.

For more information, see Controlling Trend-Data Caching in *Deploying and Managing Apama Applications* for more information.

Using trend charts

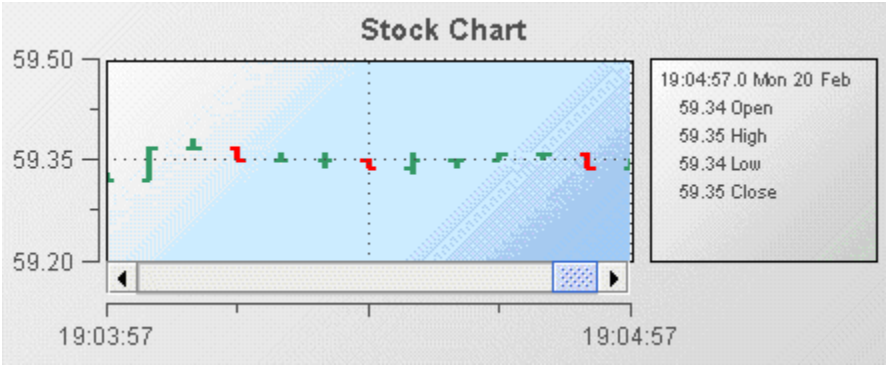
Using stock charts

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.

Common tasks related to stock charts are covered in the following sections:

"Using OHLC values" on page 110

"Creating a scenario stock chart" on page 115

"Adding overlays" on page 118

"Generating OHLC values" on page 124

Detailed reference information on stock charts is provided in "Trend Graphs" in the *Apama Dashboard Property Reference*.

Attaching Dashboards to Correlator Data

Using OHLC values

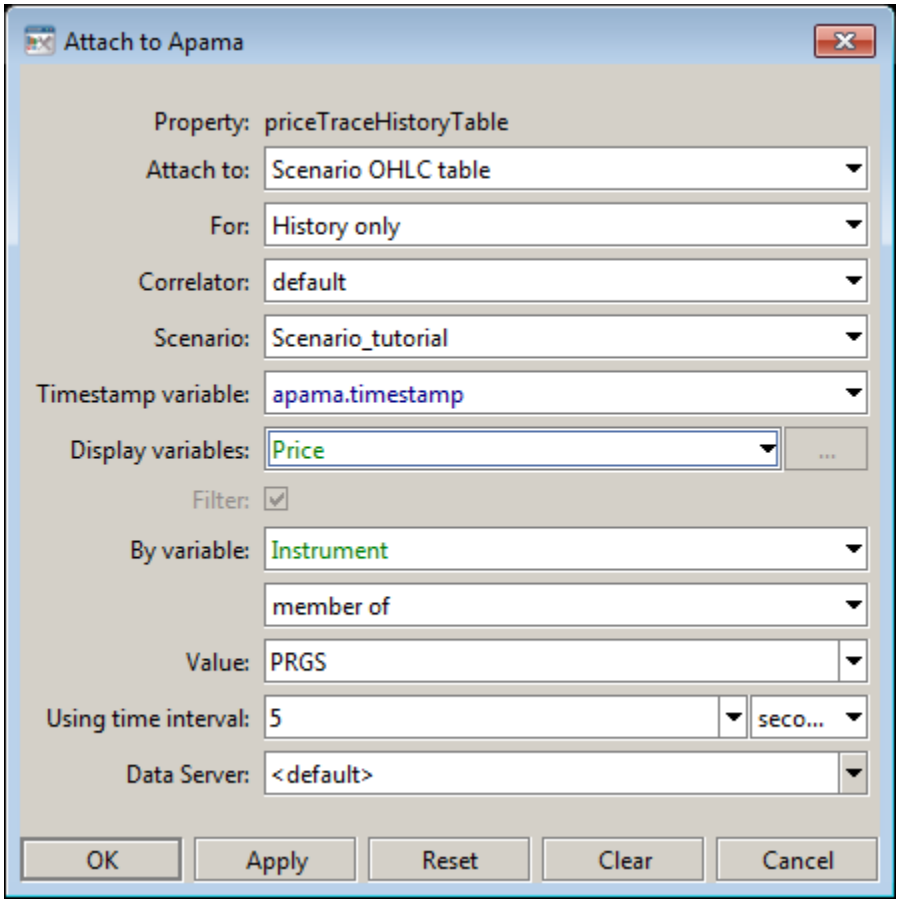
The OHLC values for a stock chart can be provided by attaching the stock chart to one of the following:

- 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)

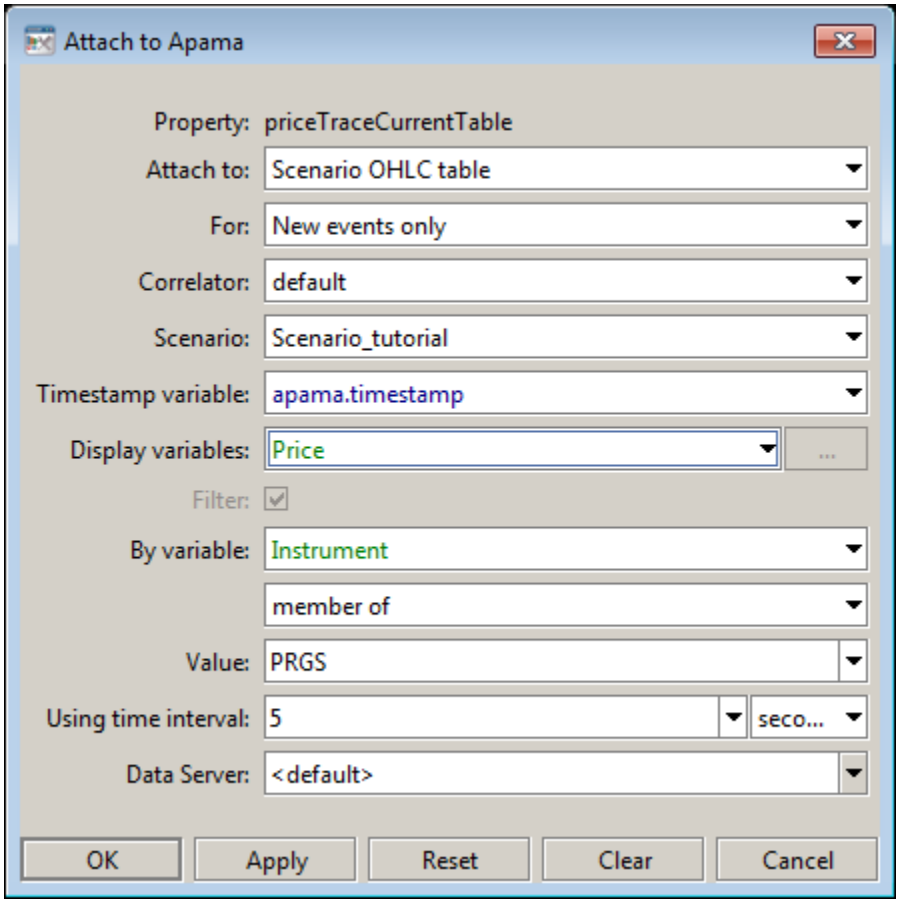
The simplest is to attach the chart to a scenario OHLC table. This is specified when creating the attachment in the Attach to Apama dialog.

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.

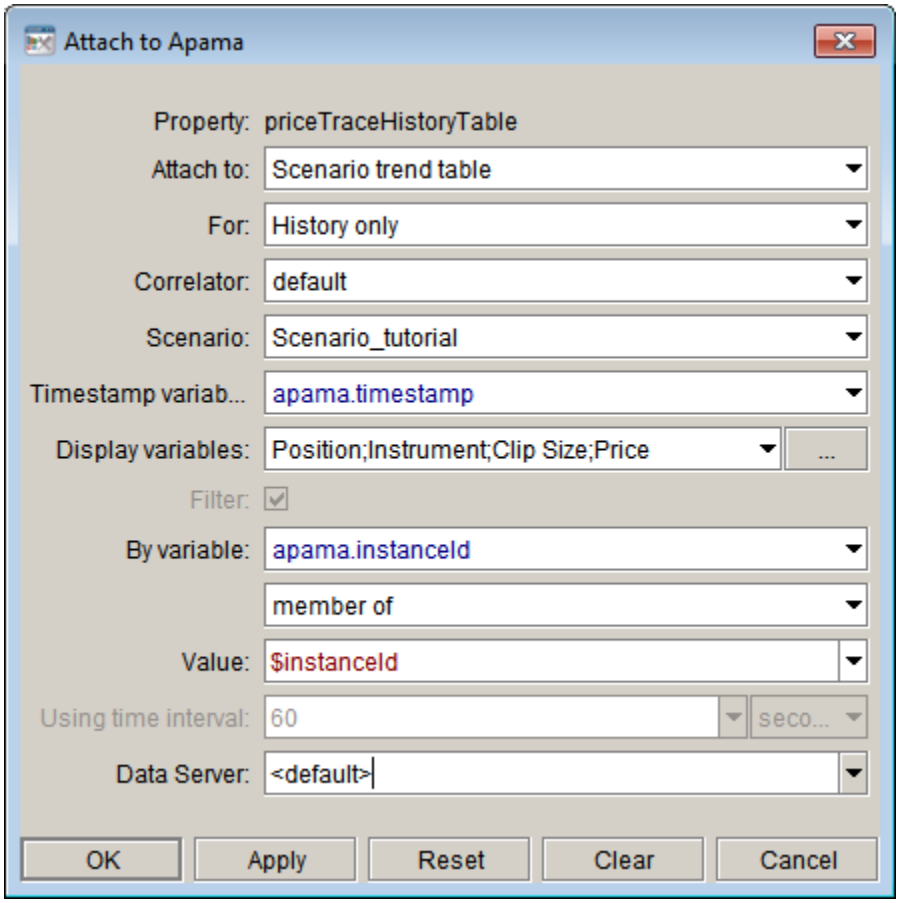


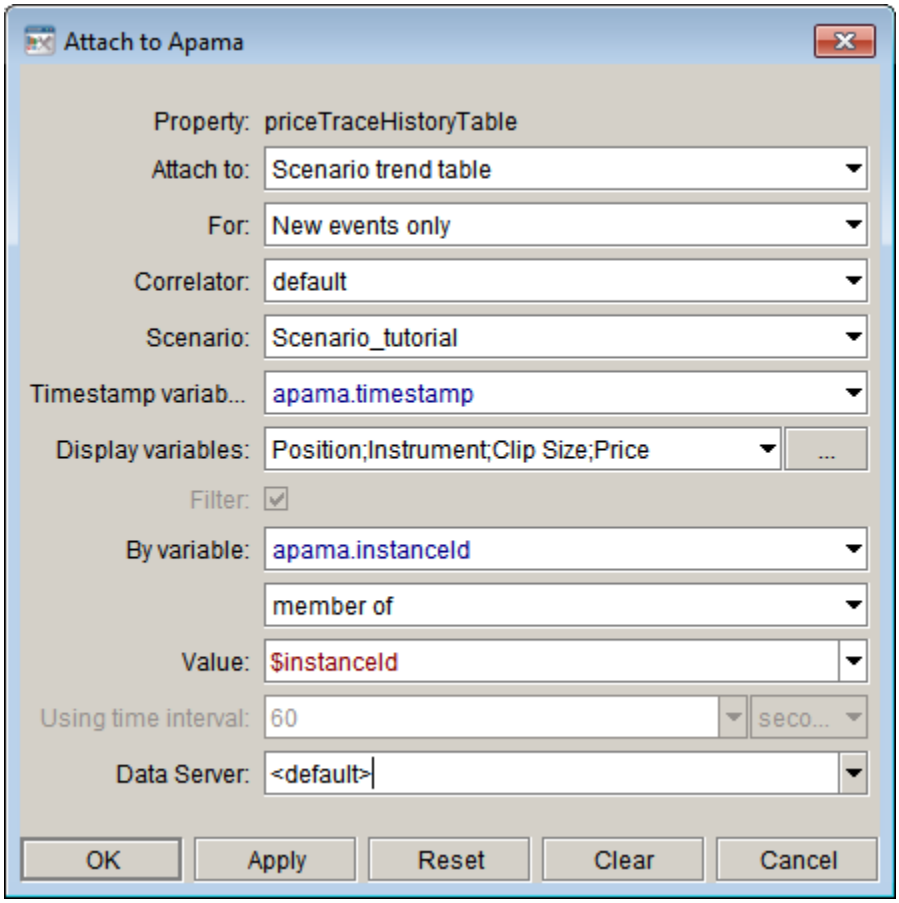
Here the attachment is made to the scenario OHLC table of the tutorial scenario and the `Price` variable is being displayed. This is the variable for which OHLC values will be calculated and displayed. The event timestamp, `apama.timestamp`, is the timestamp used to determine the time of events. The 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

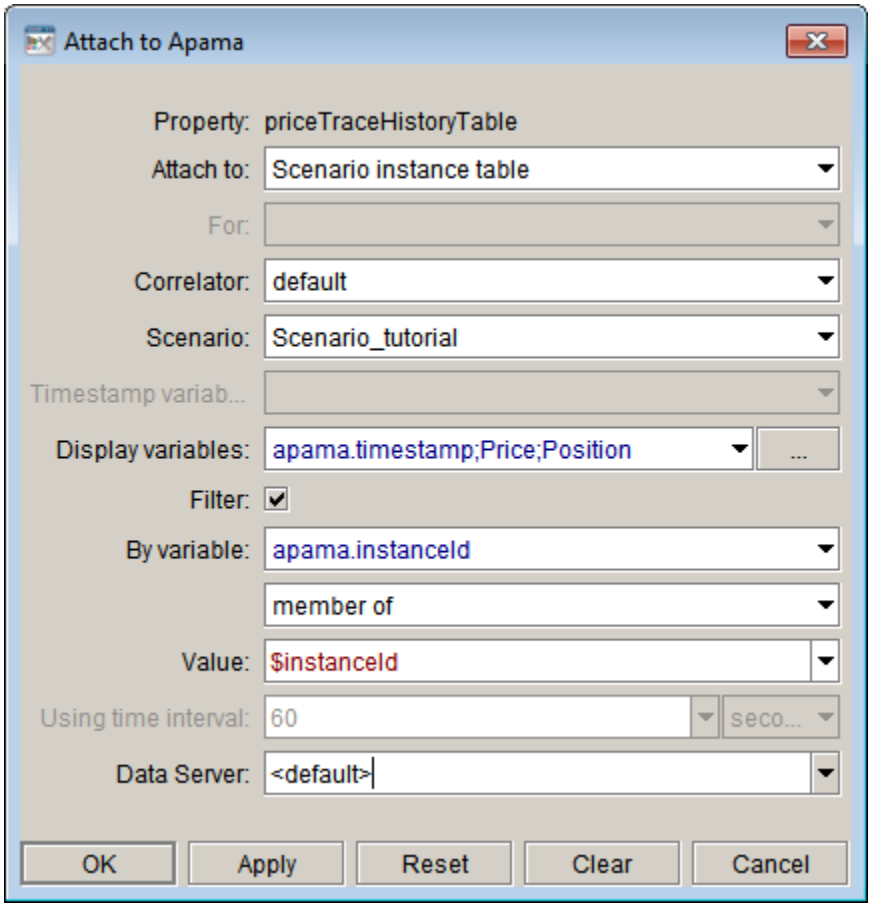


In this illustration, the attachment is made to the scenario trend table of the OHLC scenario. The scenario variables open, high, low, and close are used to provide the OHLC values. Notice that the Using time interval field is disabled. This is because the scenario is calculating the OHLC values; not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

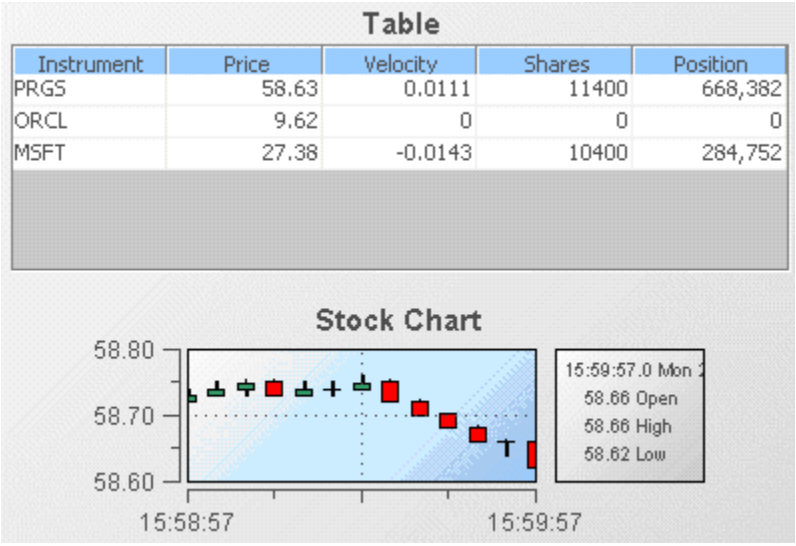
Note: Unless you have severe memory constraints or are displaying OHLC values for only a single scenario instance, you should attach the `priceTraceHistoryTable` property to either a scenario OHLC table or a scenario trend table, as this provides the best usage experience for the dashboard user.

Using stock charts

Creating a scenario stock chart

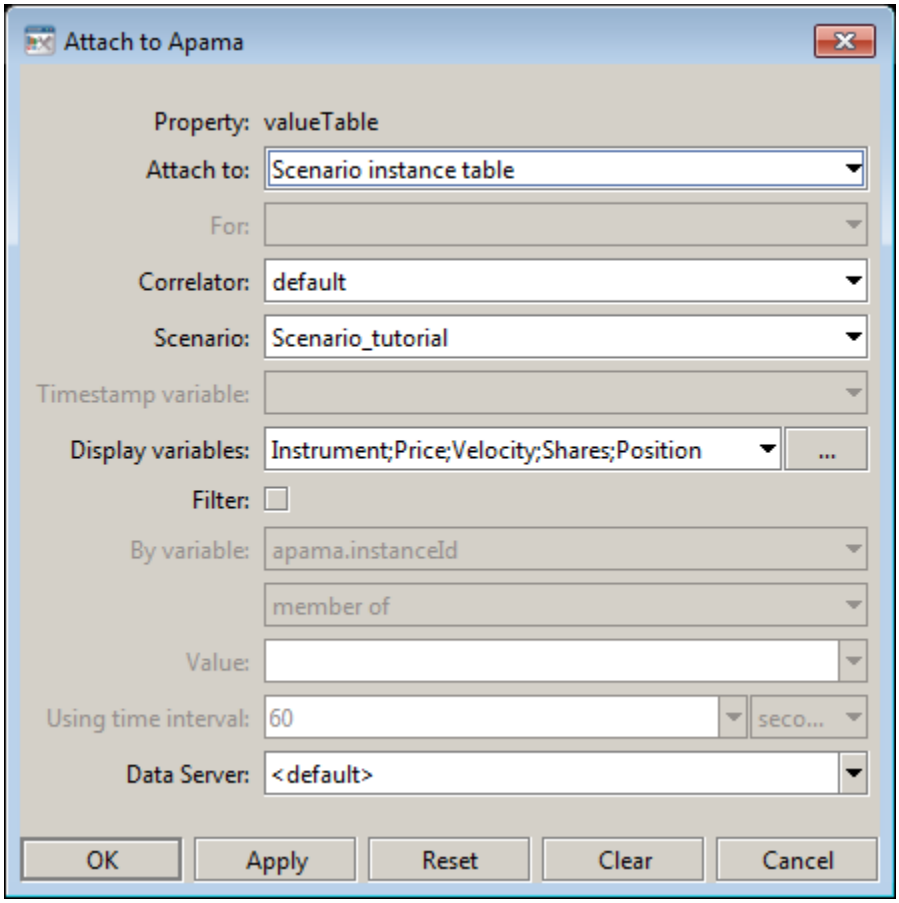
To create a stock chart for a scenario, you add it to a dashboard and attach its `priceTraceHistoryTable` property to OHLC data for a scenario instance. The filter to identify the instance will typically match on `$instanceId` although other filters could also be used.

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.



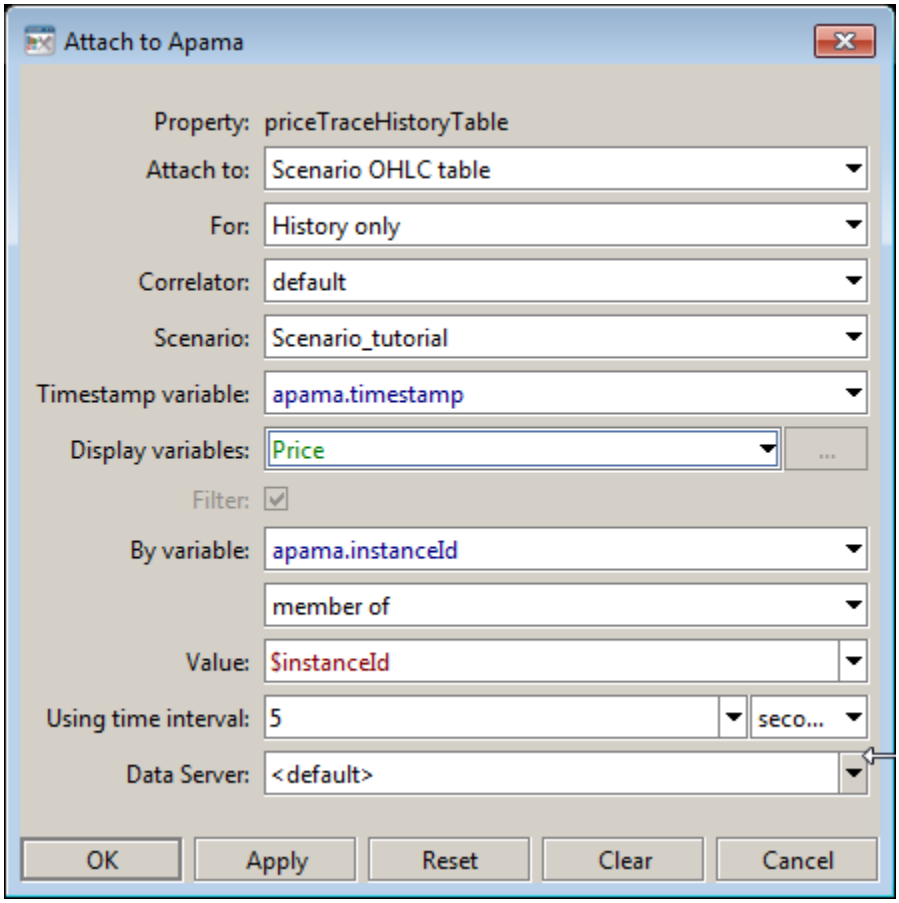
To recreate this sample create a new dashboard and perform the following steps.

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:

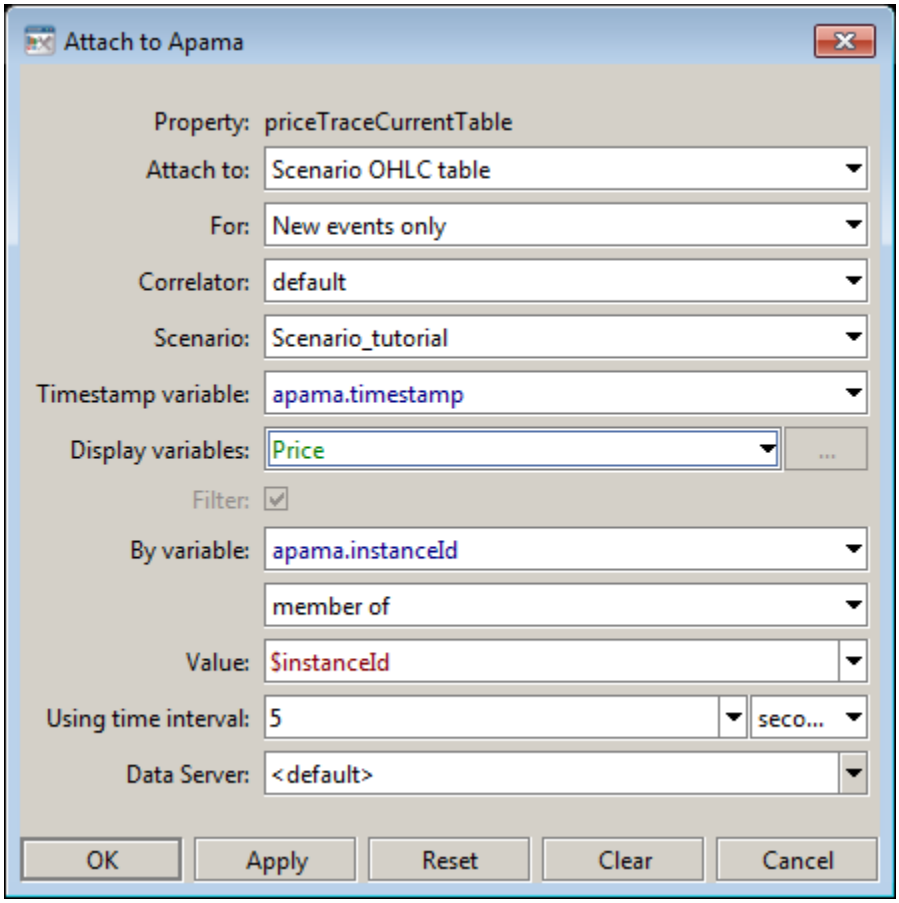


3. From the Trends tab in the Object Palette, select the Stock Chart object and add it to the dashboard Canvas.

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.



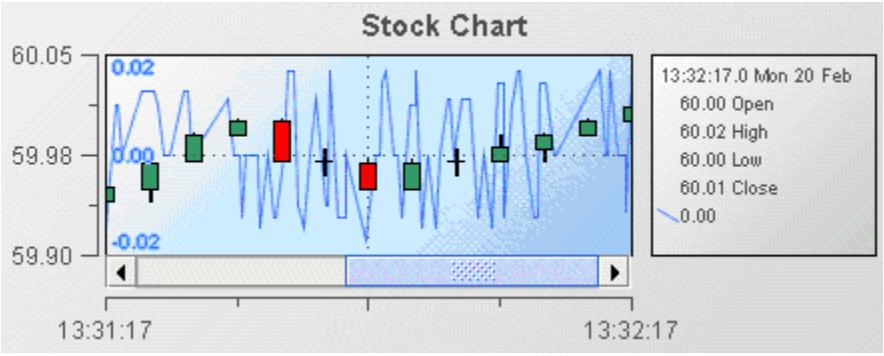
6. Select the `timeRange` property and set its value to `60.0`. This will set the chart's time axis such that 60 seconds of data will be visible. If you set the value too high you may encounter problems where the "sticks" of the chart are close and overlap.
7. Select the `scrollbarMode` property and 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.

Using stock charts

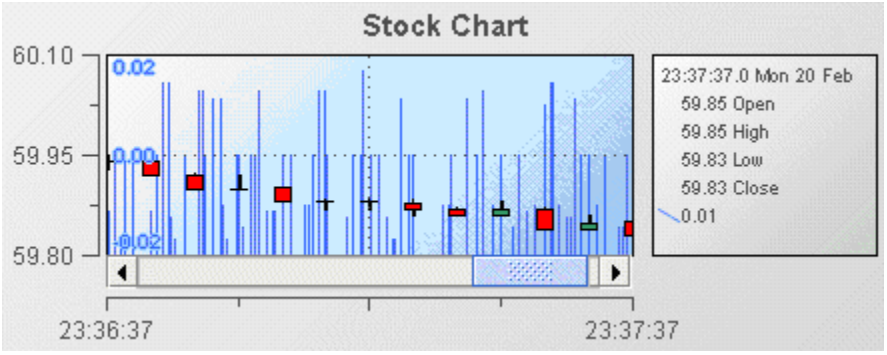
Adding overlays

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.



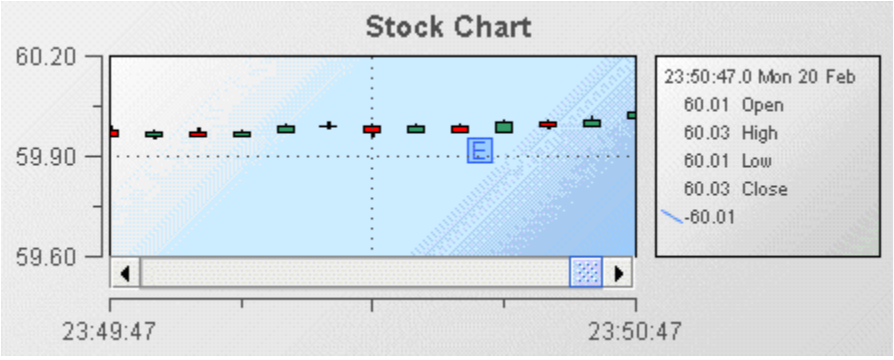
Here the overlay is showing the velocity of the stock price. Notice that multiple scales are shown on the Y axis; the outer scale corresponds to the stock price and the inner scale the velocity.

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.

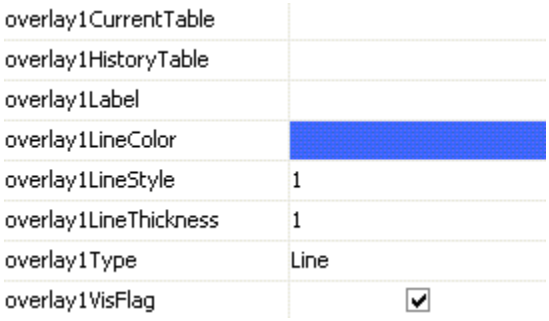
If you set `overlay1Type` to `Event`, event markers will be placed on the chart at the occurrence of each event. This allows you to easily identify when key events occurred. The following illustration demonstrates this.



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`.

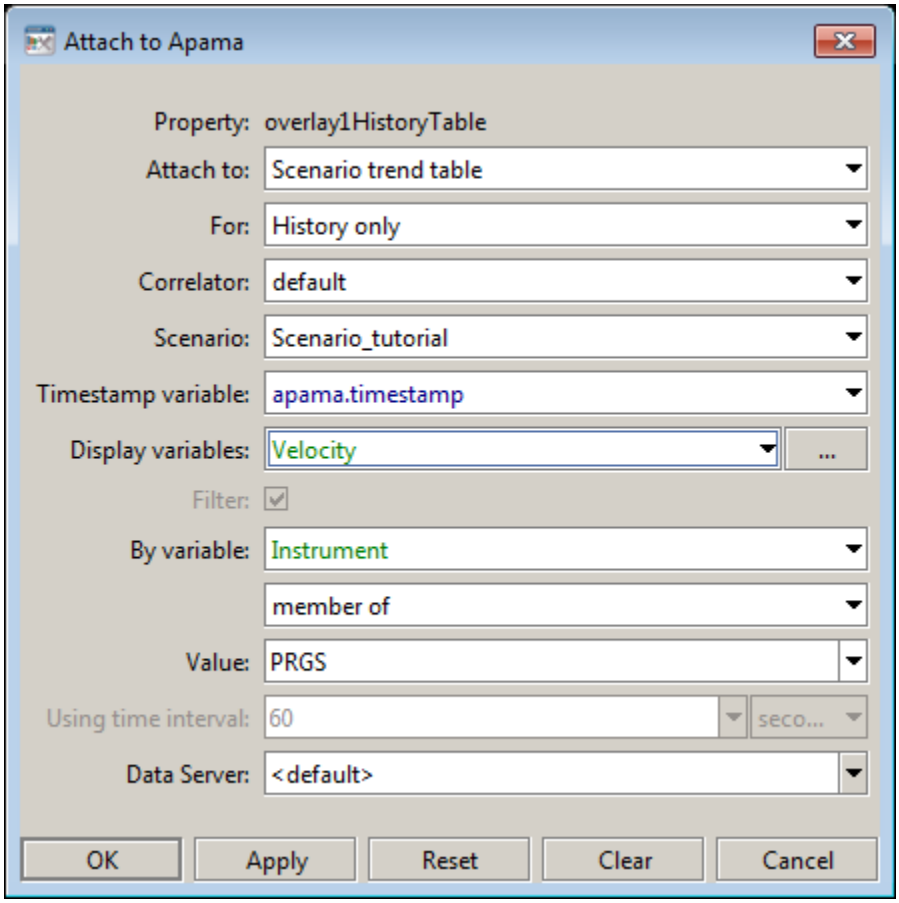
Use only the `Overlay`*n*`CurrentTable` property when attaching the overlay to a scenario instance table. Attaching to a scenario instance table requires less memory but the resulting overlay may be missing one or more data points. This can occur if the dashboard is running on a heavily loaded system. 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.

Using stock charts

Recreating the Stock Chart Overlay sample

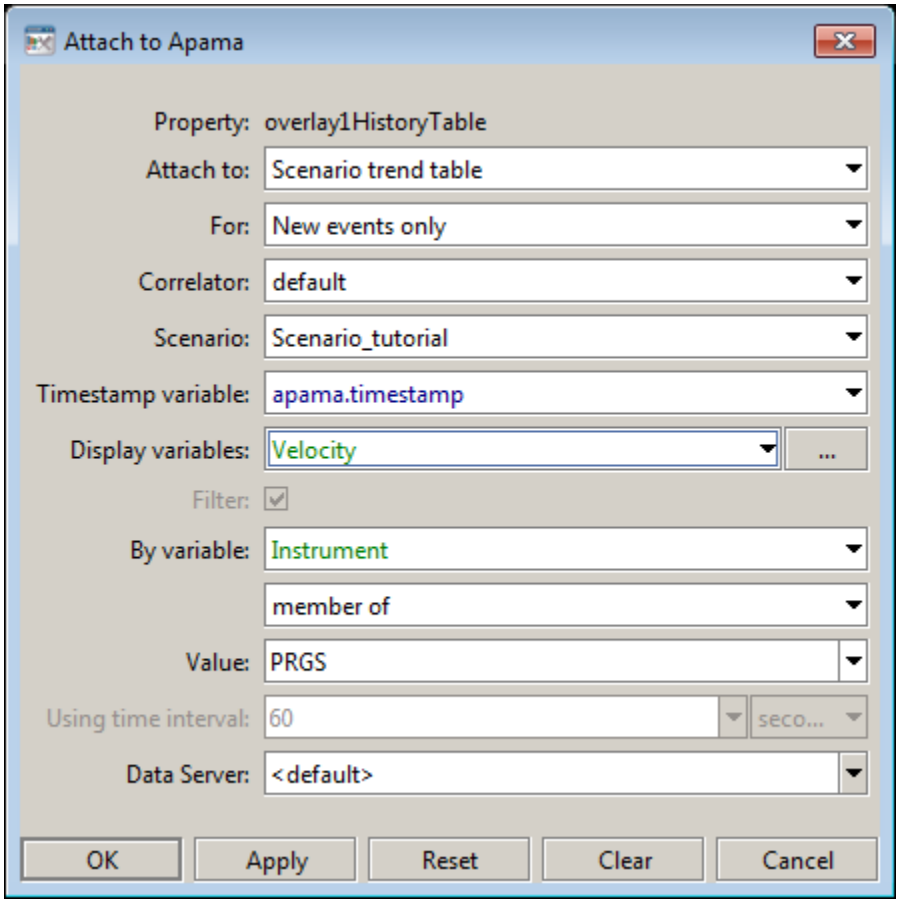
To recreate the Stock Chart Overlay sample:

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 on 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.

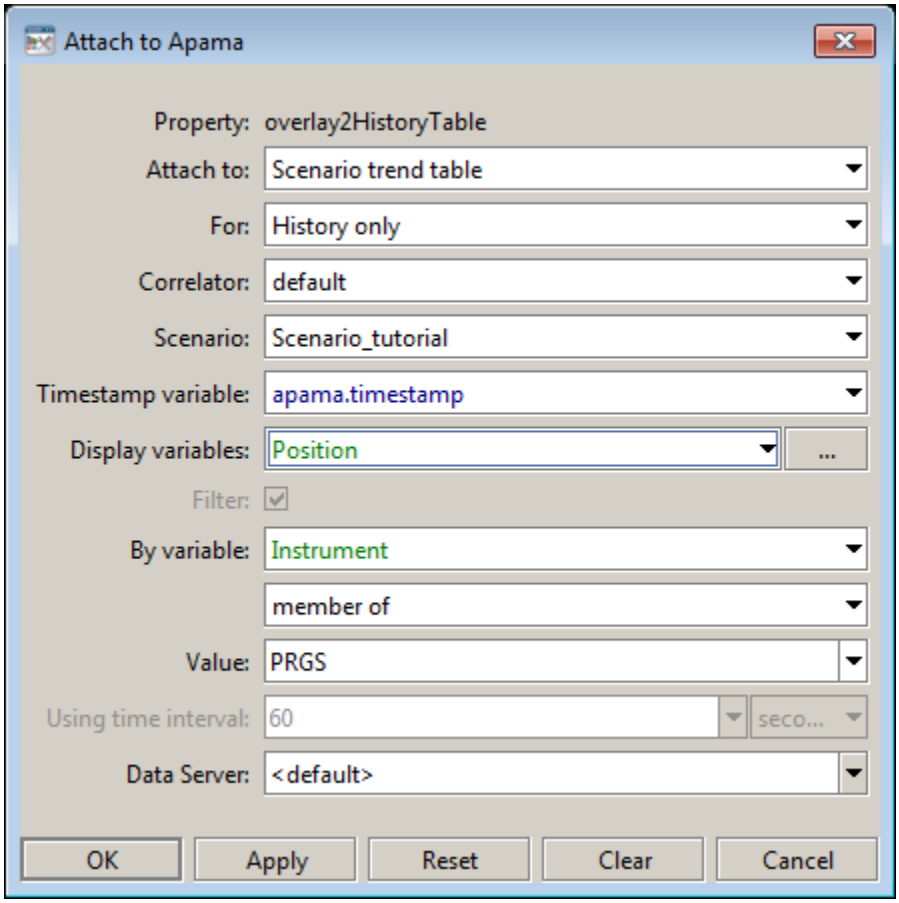
4. Double click on the `overlay1CurrentTable` property and attach it to a scenario OHLC table by specifying the following information.



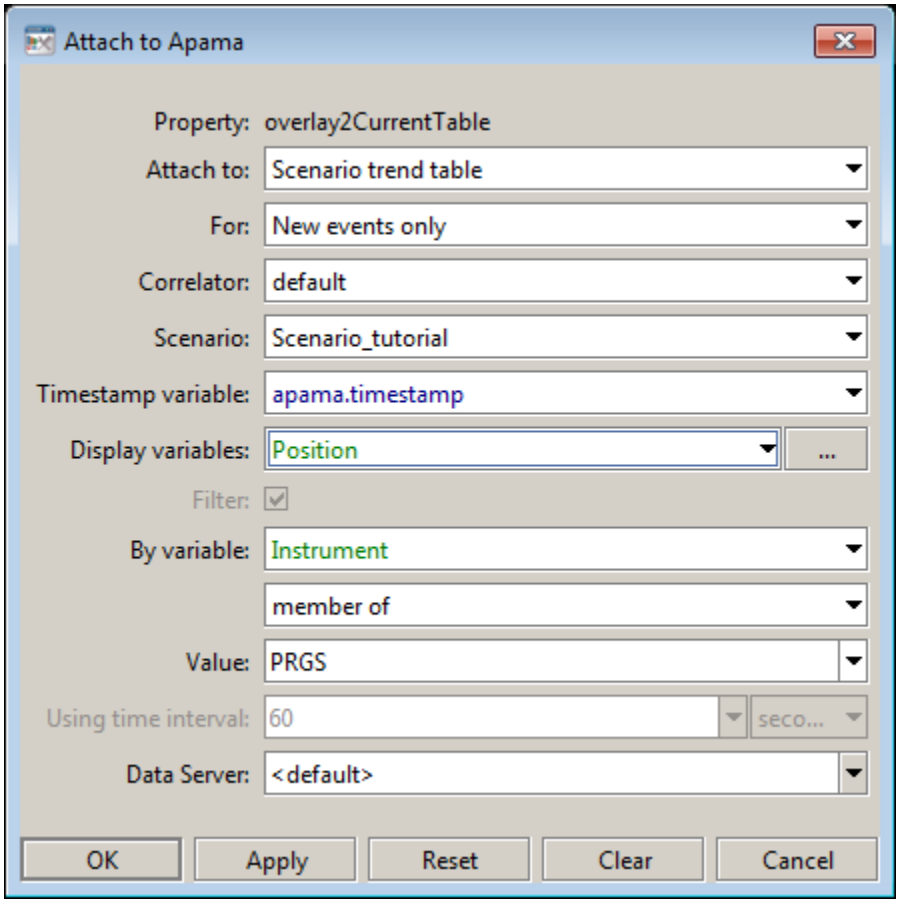
The overlay is now attached to `Velocity` property of the scenario instance where the `Instrument` equals `APMA`; this is the same filter used for the `priceTraceHistoryTable` attachment.

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.

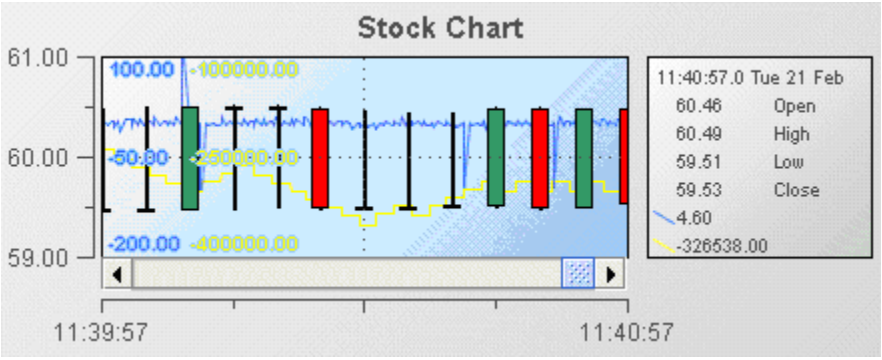
6. Double click on the `overlay2HistoryTable` property and attach it to a scenario trend table by specifying the following information:



7. Double click on 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.

Adding 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.

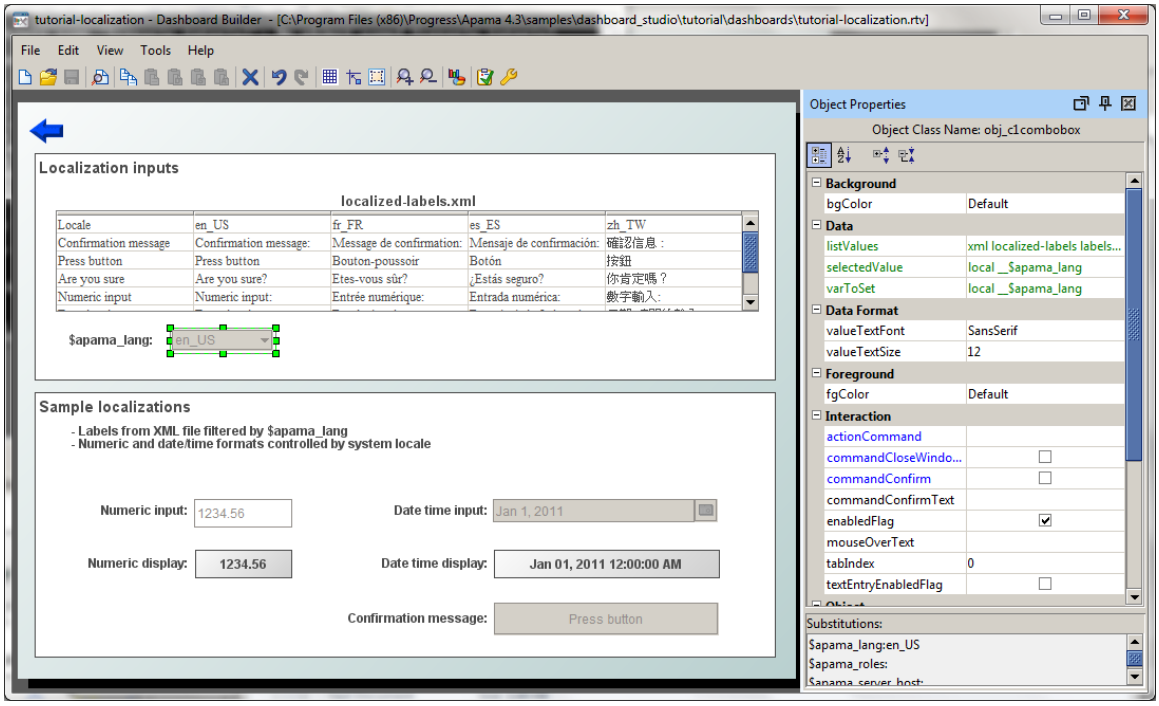
To update the OHLC variables in a single update event your scenario or DataView needs to set the value of each in the scope of a single rule. Following is an example of this in an Event Modeler rule.

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.

Using stock charts

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:



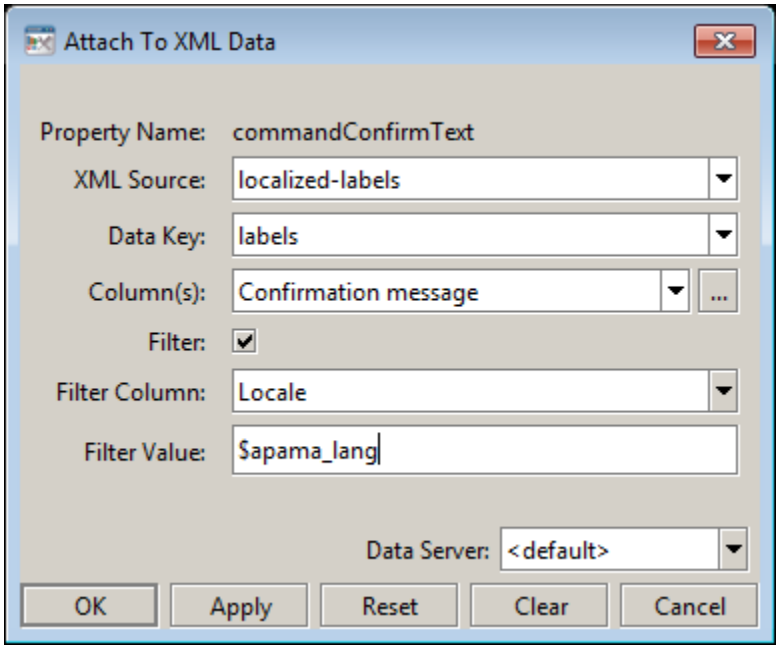
1. Create an XML dataset with a tabular data element. (See "Using XML Data" on page 228.) Create a column for supported locales, as well as a column for each label. Create a row for each locale. In each row, put a specific locale and the text for each label localized for that specific locale. Here is 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>
```

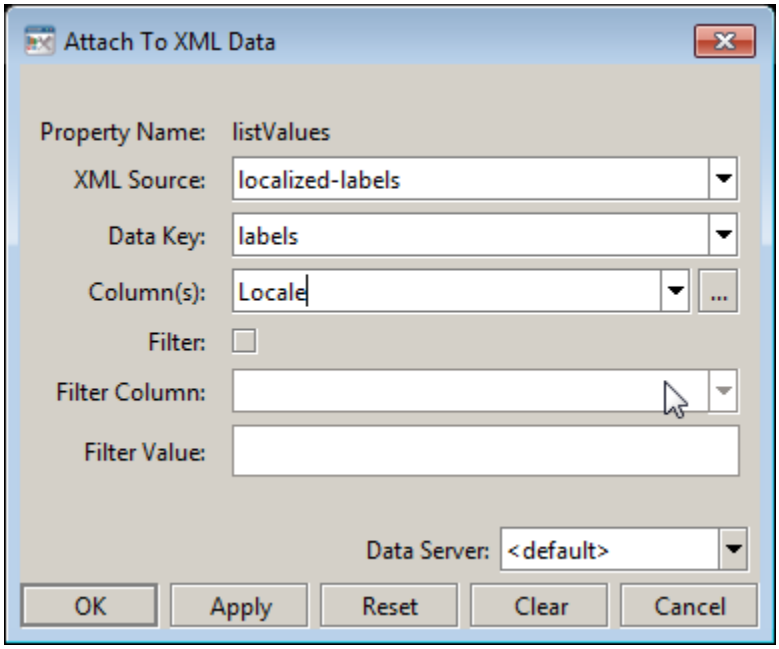
```
 <tr name="Spanish">
  <td>es_ES</td>
  <td>Mensaje de confirmación:</td>
  <td>Botón</td>
  <td>¿Estás seguro?</td>
  <td>Entrada numérica:</td>
  <td>Entrada de la fecha y hora:</td>
  <td>Exhibición numérica:</td>
  <td>Exhibición de la fecha y hora:</td>
 </tr>
 <tr name="Chinese">
  <td>zh_TW</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:

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 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.

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.

Attaching Dashboards to Correlator Data

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. Follow these steps:

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.

Attaching Dashboards to Correlator Data

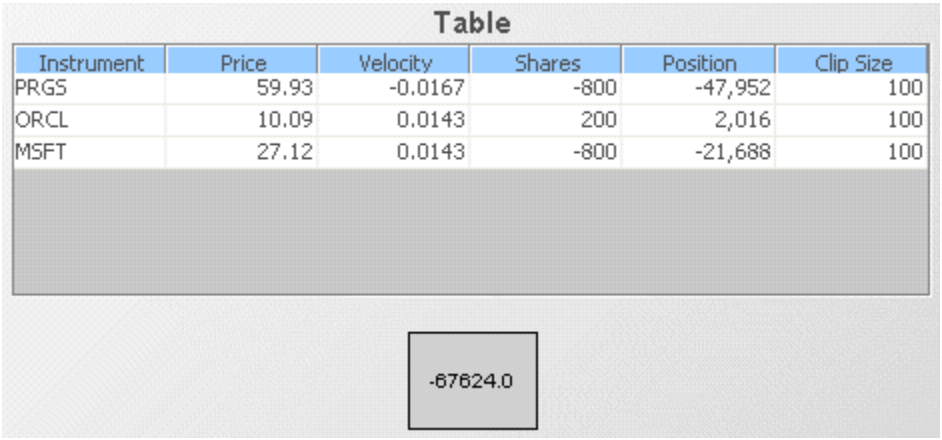
Chapter 4: Using Dashboard Functions

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

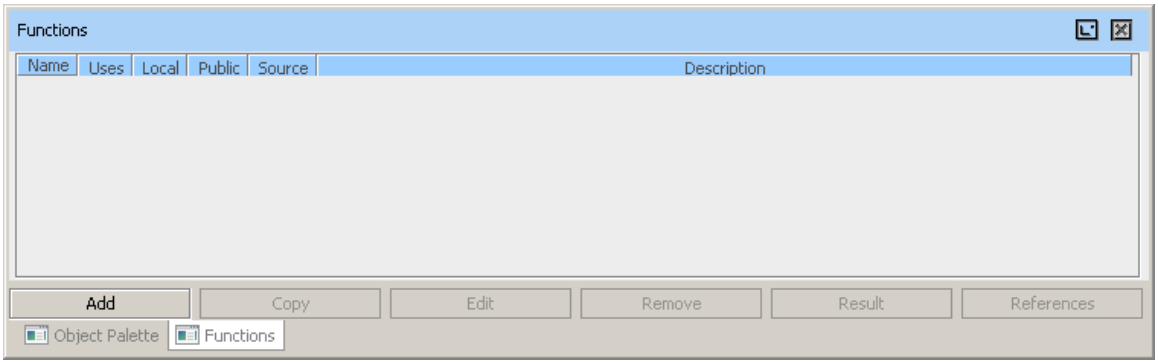
Following is an example of using a built-in function:

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.

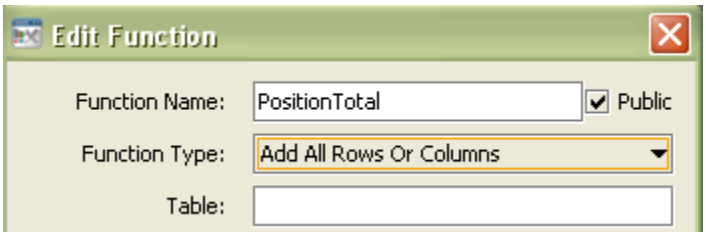
2. Open the file `tutorial-summary-table.rtv` by selecting Summary Table on the tutorial main page.
3. From the Tools menu select the Functions item to display the Functions panel.



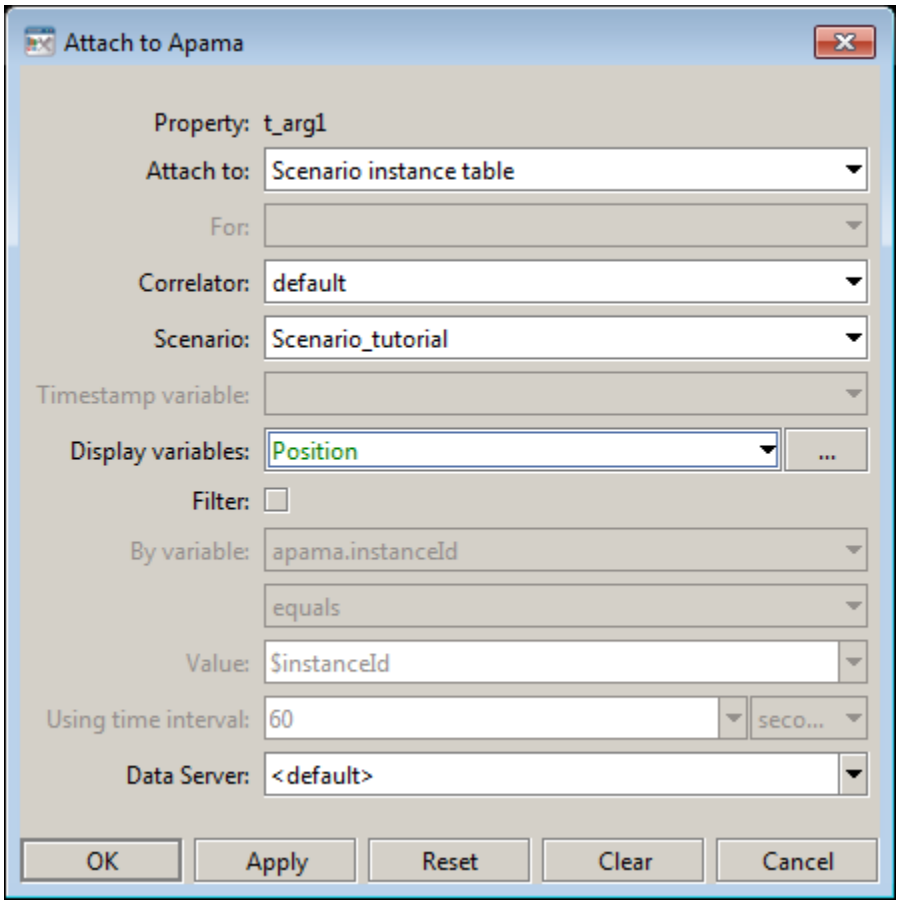
4. Click on the Add button to display the **Edit Function** dialog.

5. Set the Function Name field to `PositionTotal` and the Function Type field to `Add All Rows Or Columns`.

   For information on all builtin functions, see the Dashboard Function Reference in *Developing Apama Applications*



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



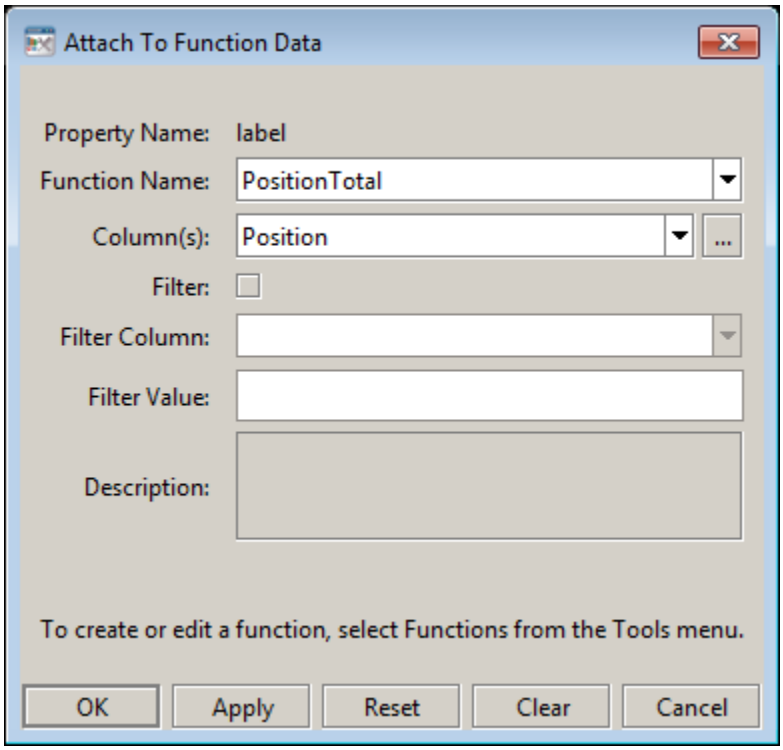
   Here the attachment specifies that the `Position` column for the tutorial scenario is to be used. The Sum function will produce the sum of the values in all the cells of a given column; in this case the sum of the cells in the Position column for all instances of the scenario.

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" in the *Dashboard Function Reference*.

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.

Using Dashboard Functions

Creating custom functions

To provide a library of functions, do both the following:

1. Develop an implementation of `com.apama.dashboard.function.IFunctionLibrary`. See "Developing a custom-function library" on page 133.
2. Install your implementation. See "Installing a custom-function library" on page 134.

Using Dashboard Functions

Developing a custom-function library

A sample implementation of `IFunctionLibrary` is included below in "Sample IFunctionLibrary implementation" on page 135.

- 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 133.
- `evaluateFunction`: Returns the result of executing a specified function with specified arguments. See "Implementing evaluateFunction" on page 133.

When you compile your implementation, ensure that `dashboard_client5.2.jar` is on your class path. This `jar` file is in the `lib` directory of your Apama installation.

Creating custom functions

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 135.

Developing a custom-function library

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 `dashboard_client5.2.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);
```

Developing a custom-function library

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 Apama Studio, right click on your project and select Properties, expand Apama, select Dashboard Properties, activate the External Dependencies tab, and click the Add External button).

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.

Creating custom functions

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;
  }
}
```

Creating custom functions

Chapter 5: Defining Dashboard Commands

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 220), as is defining SQL commands (see "Using SQL Data" on page 234).

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 Dashboard Commands

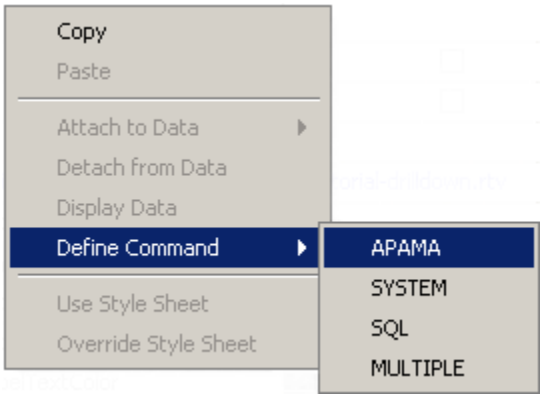
Defining commands

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.

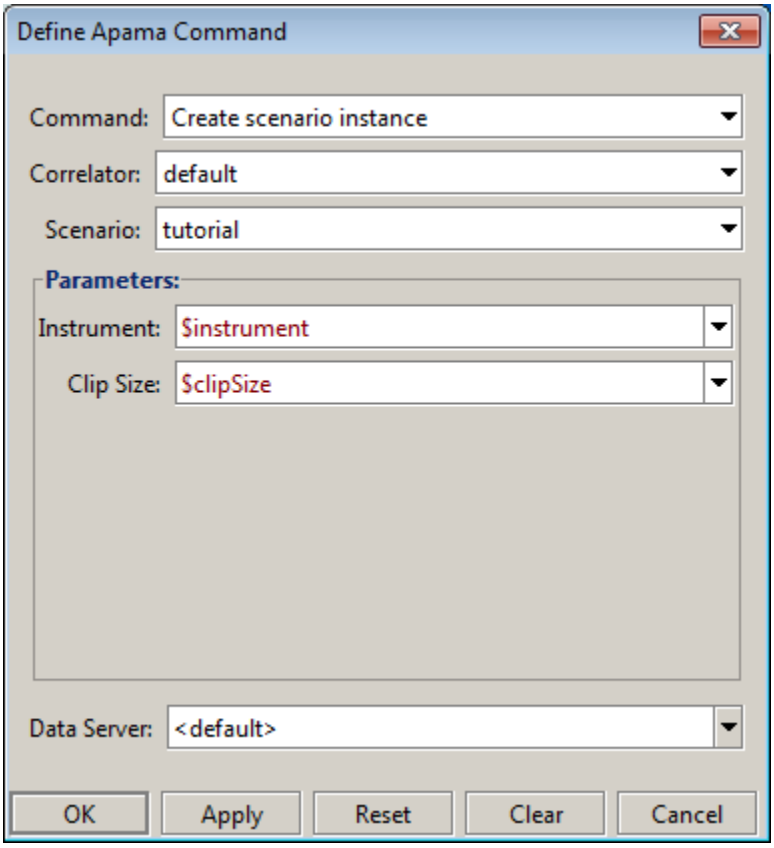
For control objects such as push buttons, commands are defined by setting the `actionCommand` property. For other objects such as labels and charts, the commands are defined by setting the `command` property.

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 on 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 70 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.

Defining Dashboard Commands

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.

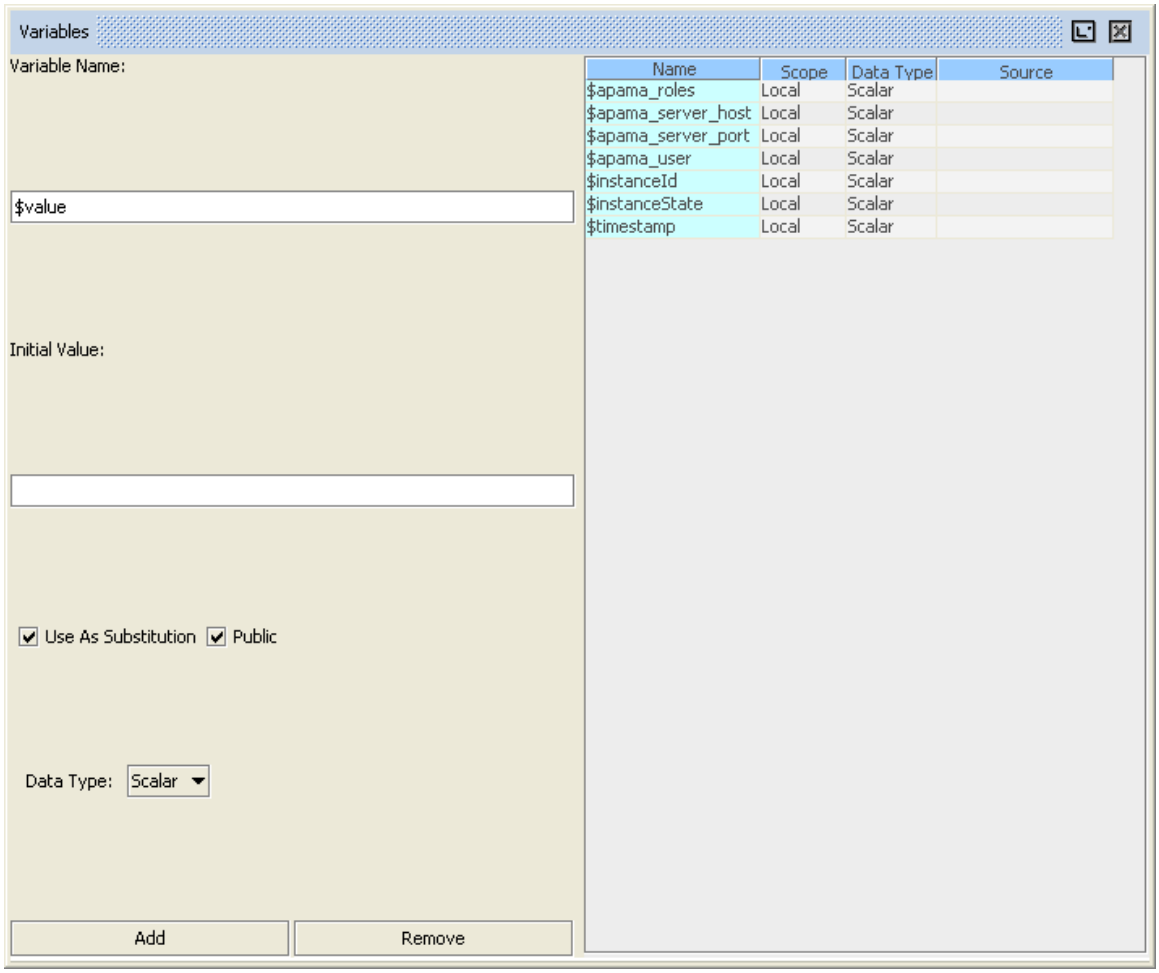
For an example of how this works:

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.

   The resulting dashboard should look similar to the following.

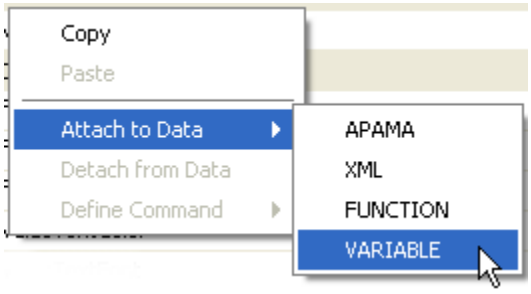
   You will now associate the text field with a variable so that when its value changes the label object updates to show the value.
4. Add the dashboard variable `$value` by selecting Variables from the Tools menu and adding it in the Variables panel.



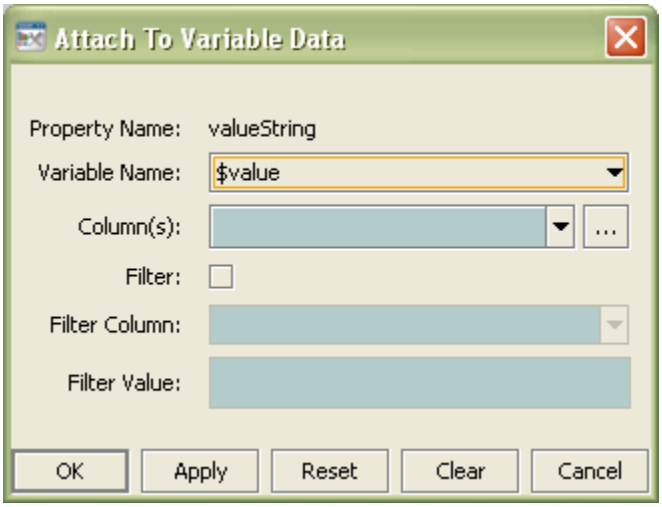
5. Ensure that Use As Substitution is checked. Be sure to click on the Add button to add it the list of variables.

   Several substitution variables are automatically created when you create a dashboard.

6. Select the label object and in the Object Properties panel, right click on the `valueString` property and select Attach to Data | Variable.



   This displays the **Attach to Variable Data** dialog.



7. Select `$value` and click on 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 Dashboard Commands

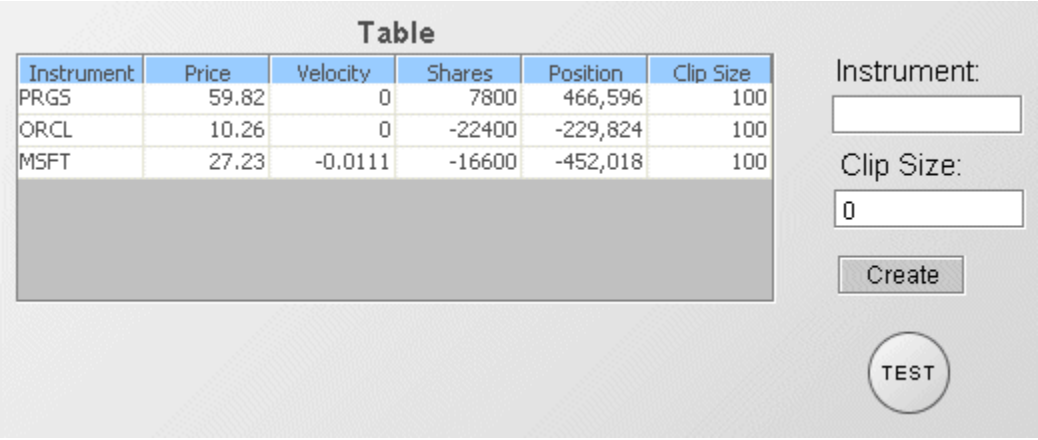
Defining commands for creating a scenario instance

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.

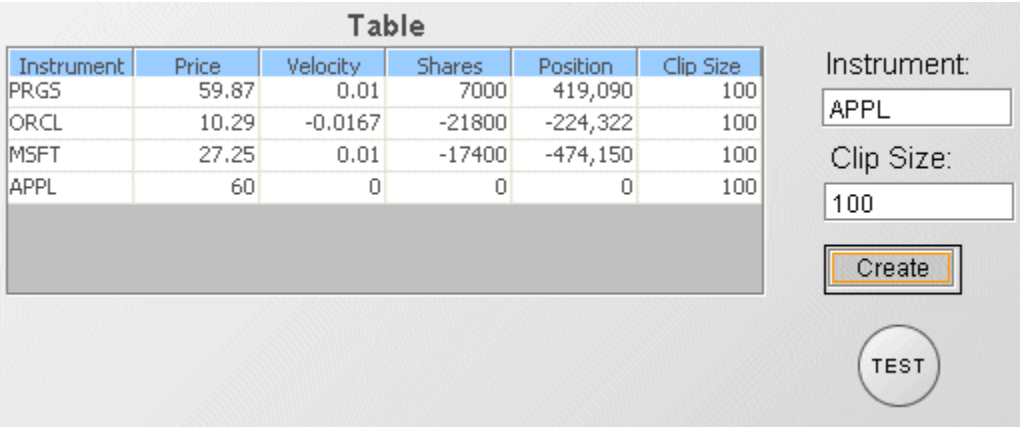
1. Open the file `tutorial-create.rtv` by selecting Create Instance in the tutorial main page.

2. Double click on 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.

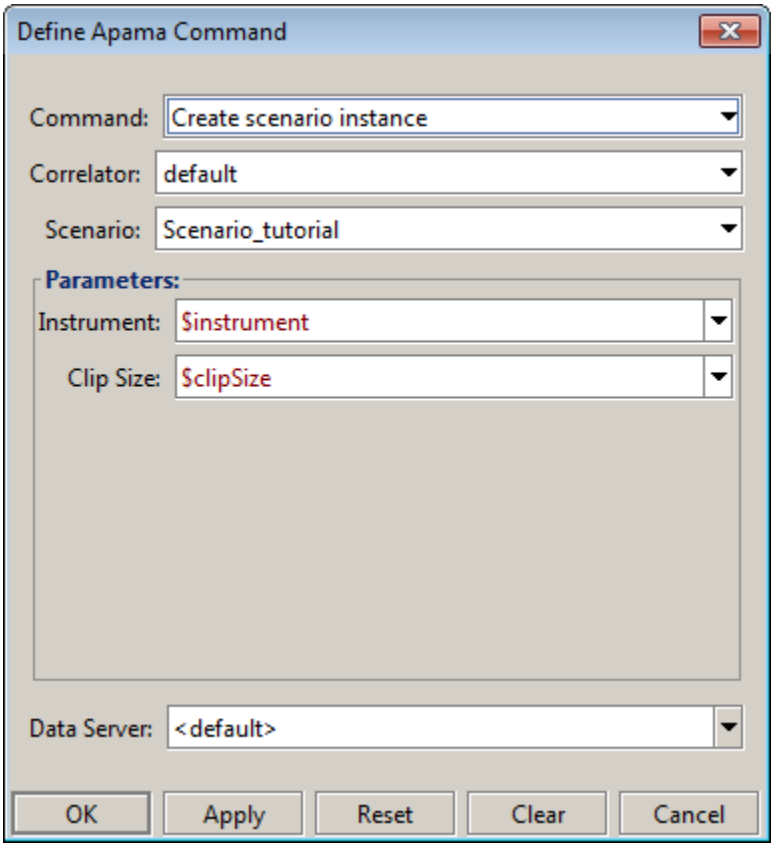
3. In the form enter `APPL` for the `Instrument` and `100` for the `Clip Size` and click on the Create button. This will create a new instance of the scenario..



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 on 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 Dashboard Commands

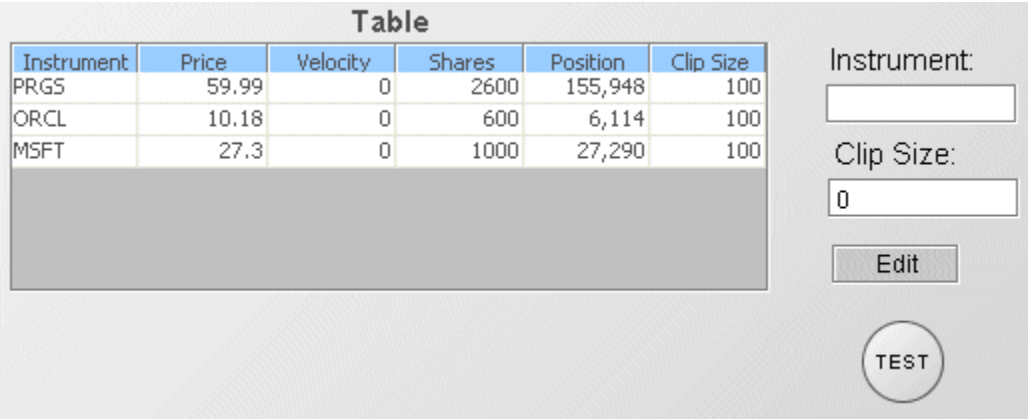
Defining commands for editing a scenario instance

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.

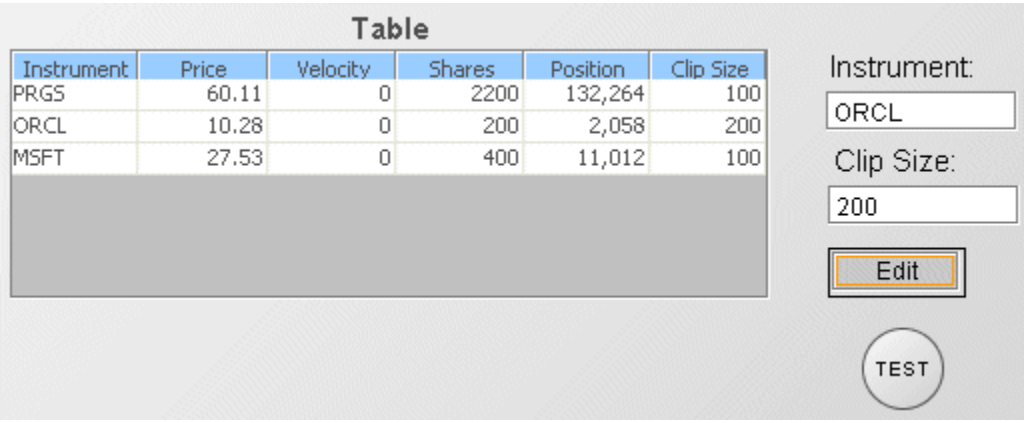
1. Open the file `tutorial-edit.rtv` by selecting Edit Instance in the tutorial main page.
2. Double click on the object labeled Test to display the dashboard in a new window such that the control objects are enabled..



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

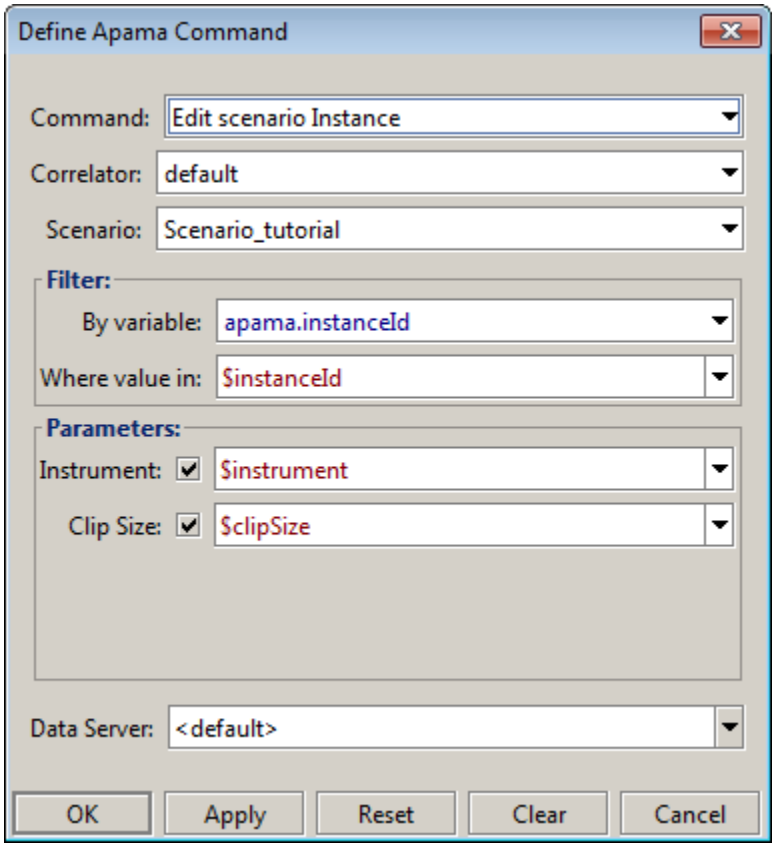
3. Double click on the `APMA` row in the table. This will cause the scenario instance for APMA to become selected and its input variables to be displayed in the form.

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 on 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.

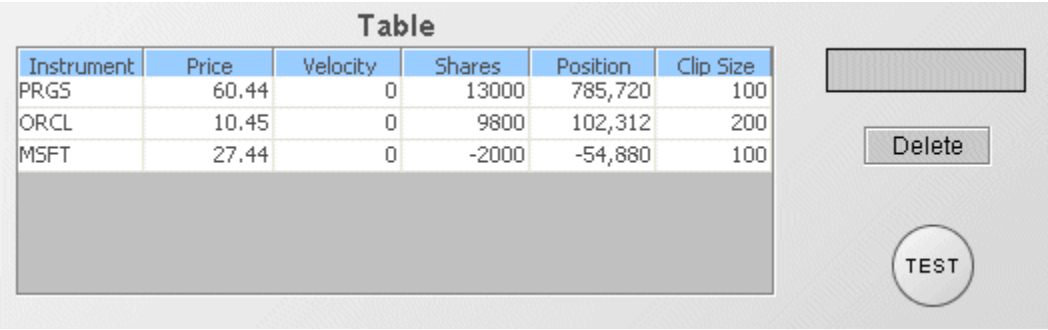
The properties of table and other objects in Dashboard Builder are preconfigured to set `$instanceId` when a drilldown is performed. However, you can use dashboard variables other than `$instanceId` to hold the `apama.instanceId` of a scenario instance.

Defining Dashboard Commands

Supporting deletion of a scenario instance

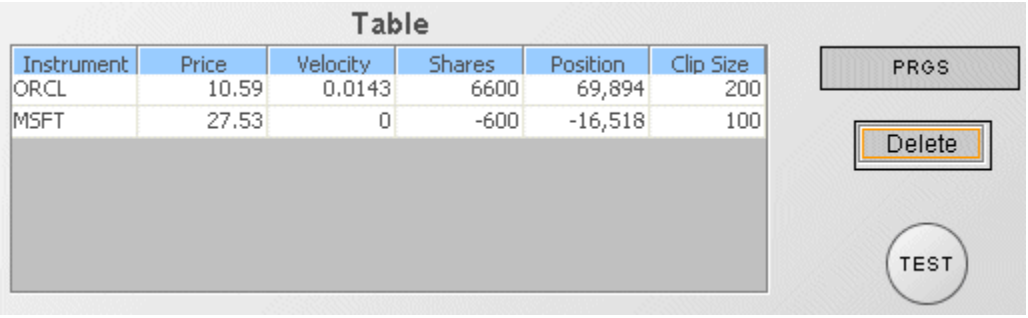
To add the `Delete` function to a dashboard you need to add a control object such as a push button and set its action to perform the delete.

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



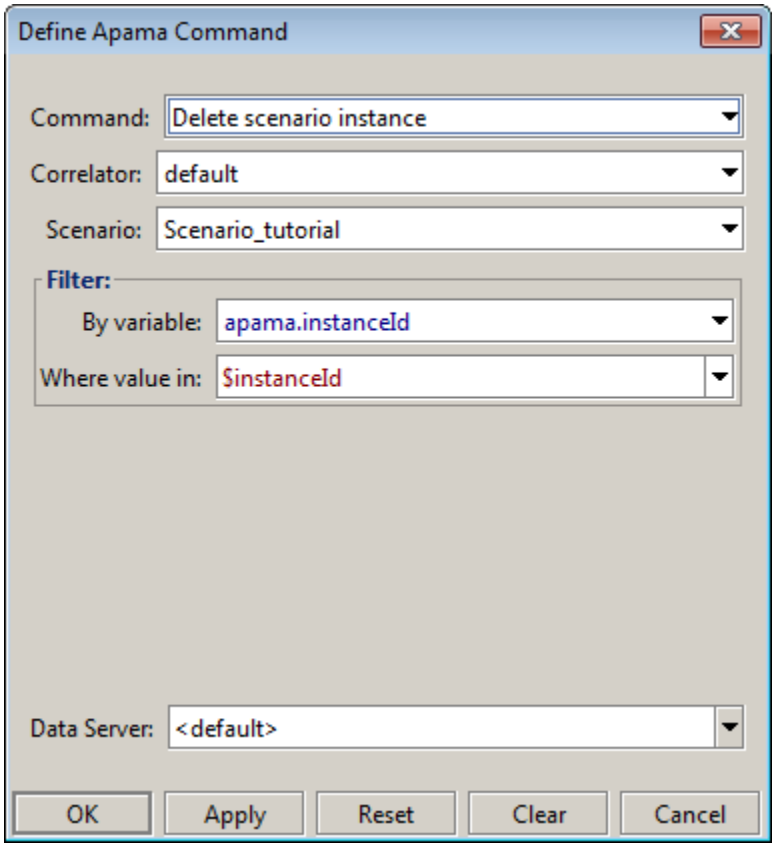
This dashboard displays a summary table of all instances of the tutorial scenario and a Delete button for deleting the selected instance.

3. Double click on 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 on the `actionCommand` property.

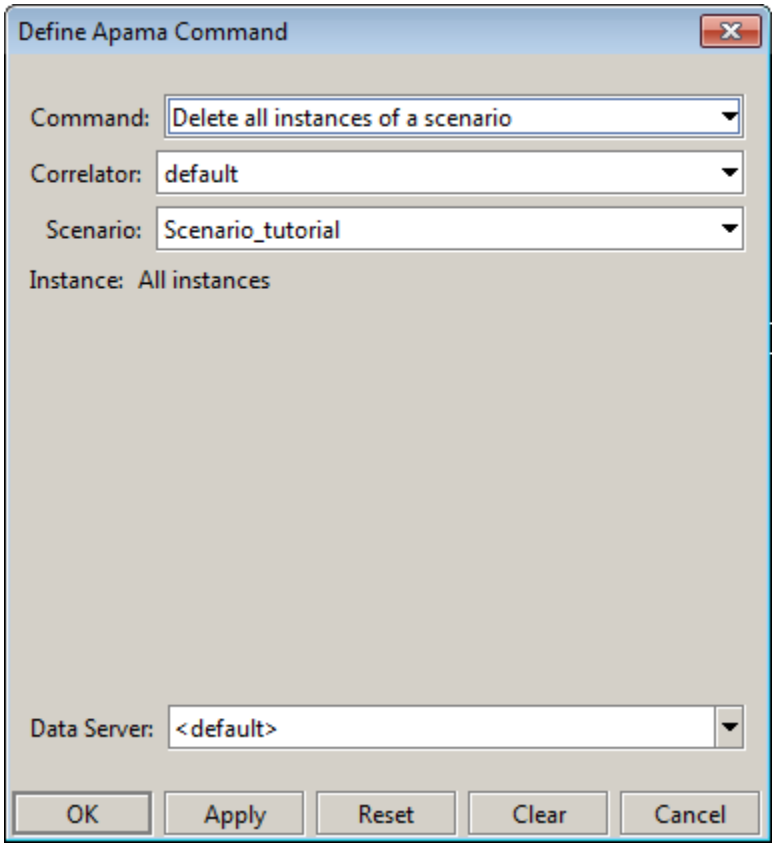


Here the command is defined to delete the instance of the tutorial scenario whose instance id equals `$instanceId`.

Defining Dashboard Commands

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.



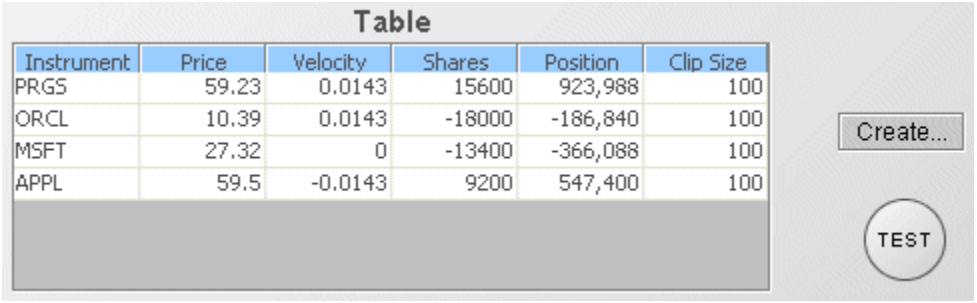
Deleting all instances of a scenario will only delete those instances to which the user has delete access. By default, these are the instances created by the user.

Defining Dashboard Commands

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.

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

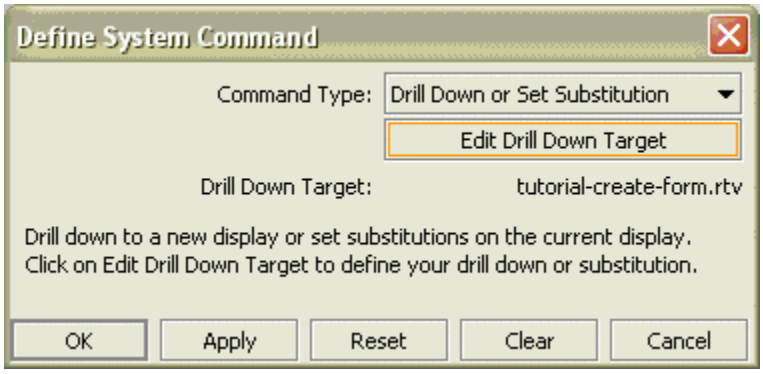


Here the dashboard contains a Create button but no fields for setting the input variables.

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

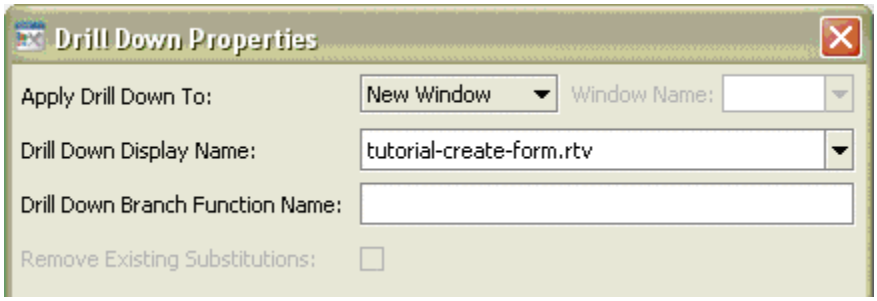
   This dialog is really just another dashboard, in this case `tutorial-create-form.rtv`. The Create button displays this dialog by performing a drilldown and displaying `tutorial-create-form.rtv` in a new window.

4. Select the Create button object and double click on 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. This gives it the behavior of a popup dialog.

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

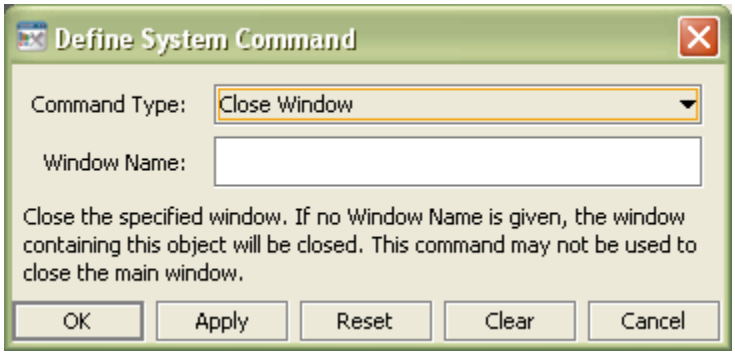
6. Open the file `tutorial-create-form.rtv`.

   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 on the `actionCommand` property..



Here the command is set to close the window.

Defining Dashboard Commands

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.

Defining Dashboard Commands

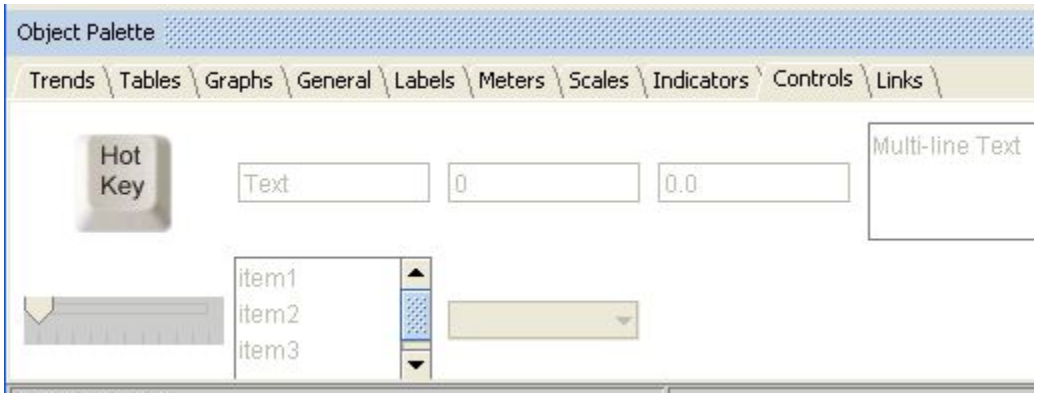
Associating a command with keystrokes

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`

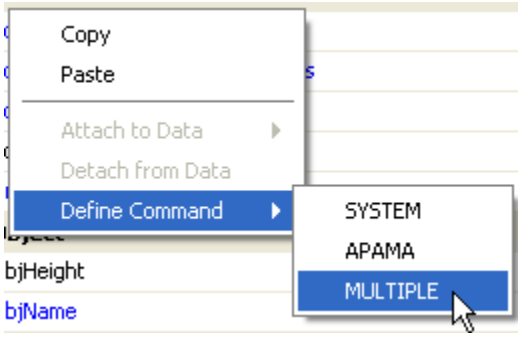
For the dashboard user, when focus is on the dashboard, the specified key sequence triggers execution of the command.

Defining Dashboard Commands

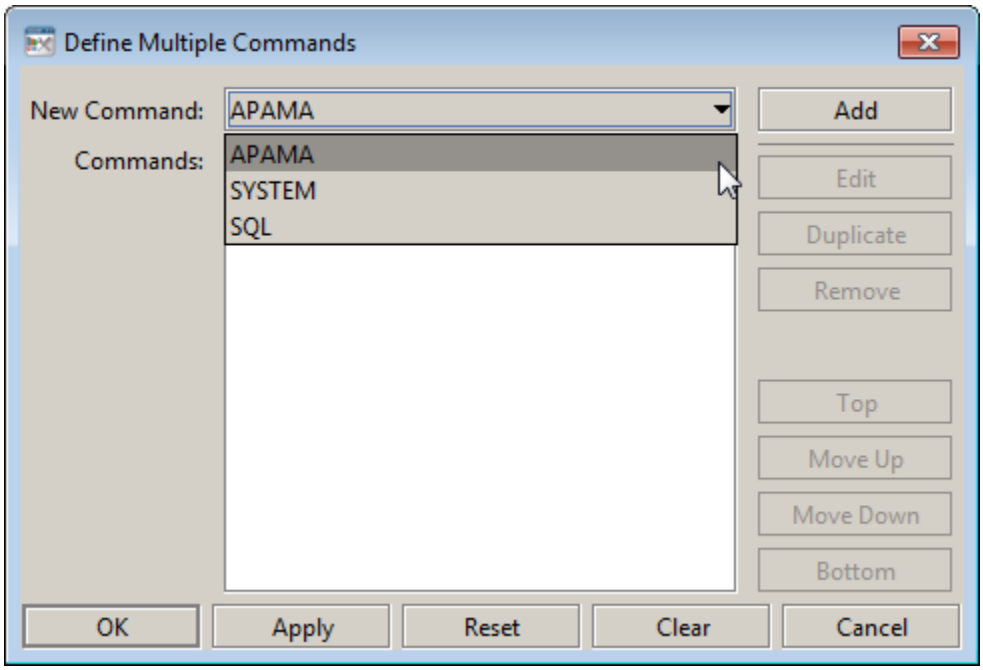
Defining multiple commands

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

1. Right click on the `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 160.

Defining Dashboard Commands

Creating custom commands

To provide a Data Server or Display Server with a library of custom commands, do both the following:

1. Develop an implementation of `com.apama.dashboard.function.ICommandLibrary`. See "Developing a custom-command library" on page 158.
2. Install your implementation. See "Installing a Custom-Command Library" on page 159.

Defining Dashboard Commands

Developing a custom-command library

A sample implementation of `ICommandLibrary` is included below in "Sample ICommandLibrary implementation" on page 159.

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 `dashboard_client5.2.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);
```

Creating custom commands

Installing a Custom-Command 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 `ICommandLibrary` implementation. The line must have the following form:

`command` *fully-qualified-classname*

- 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 Apama Studio, right click on your project and select Properties, expand Apama, select Dashboard Properties, activate the External Dependencies tab, and click the Add External button).

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.

Creating custom commands

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;
  }
}
```

Creating custom commands

Apama set substitution command

To set substitution values without using the Drill Down or Set Substitution system command, use the Apama set substitution command:

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.

Defining Dashboard Commands

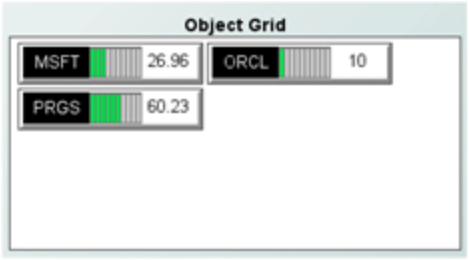
Chapter 6: Reusing Dashboard Components

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.

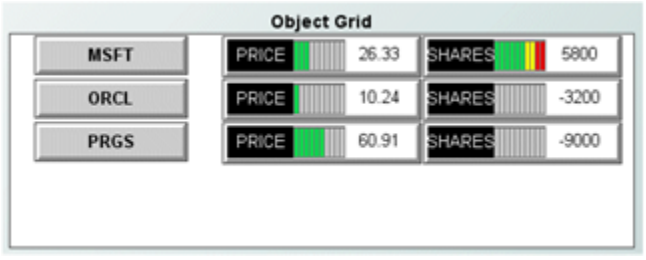
Using Object Grids

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.

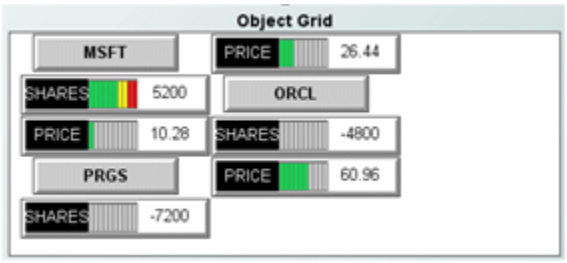


The grid above uses three objects to display the Instrument, Price, and Shares variables of the scenario instances.

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:



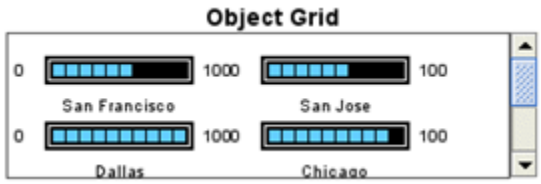
This is the same object grid as in the previous illustration. The only change is that it was resized to be slightly narrower which caused the flow layout of objects to change.

If precise control over the layout of objects is required use the Composite or Composite Grid objects.

Reusing Dashboard Components

Configuring Object Grids

The Object Grid is in the Composite tab of the object pallet.

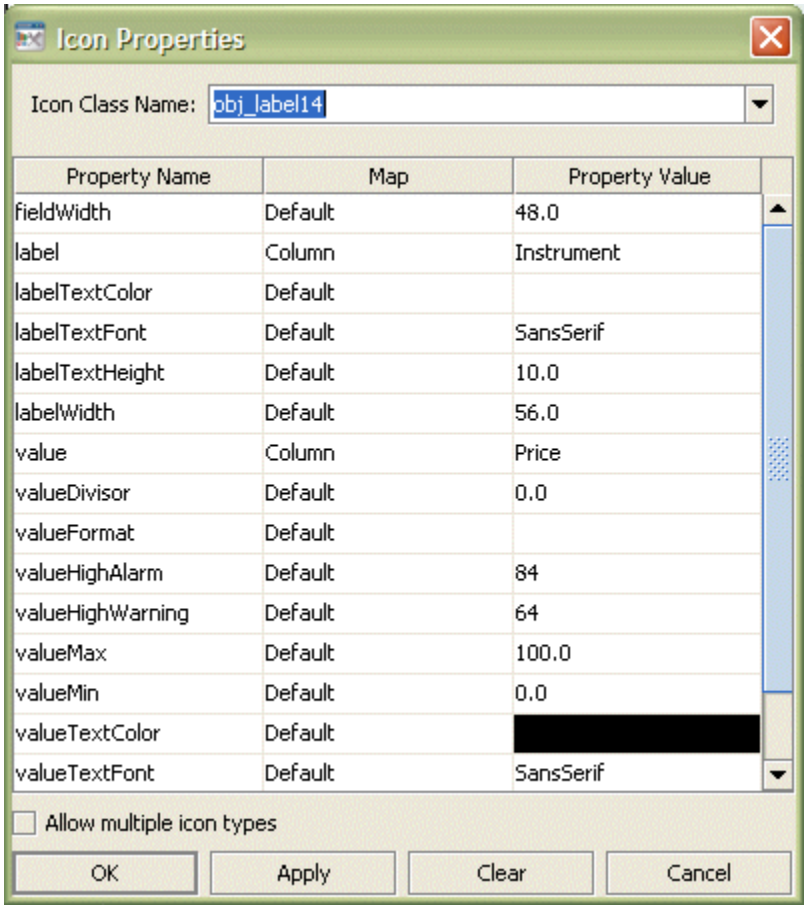


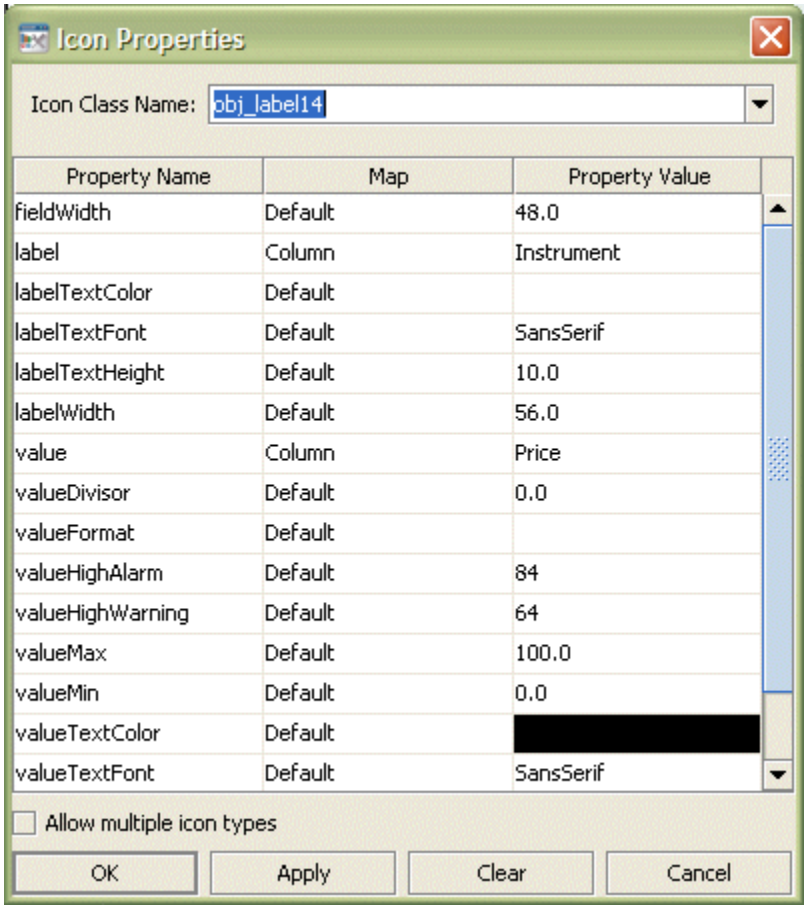
The Object Grid is initialized to display the same sample data as the Table object. The sample data contains seven rows so there are seven instances of the object in the grid.

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:

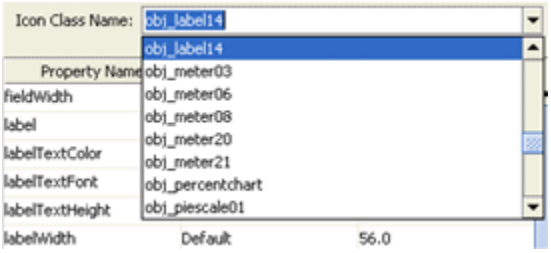
- Apama scenario instance tables
- Apama DataViews
- Dashboard functions that produce tabular data
- Tabular XML data

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 on the `iconProperties` property to display the **Icon Properties** dialog.





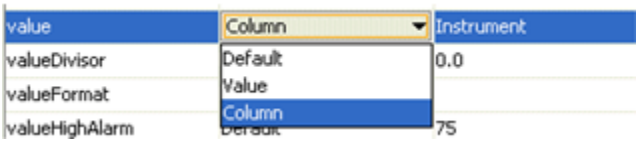
By default the Object Grid is configured to display a single object for each row in its tabular data. The Icon Class Name field is where you select the type of object that you want to display in the grid:



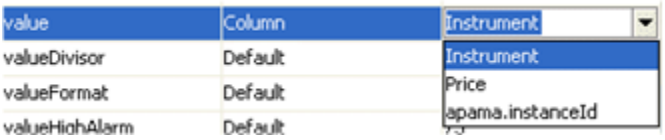
The properties listed correspond to the properties of the object type selected. Properties in the Icon Properties dialog can have their value set in one of three ways. How a property is being set is indicated in the Map column of the property list:

- 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.



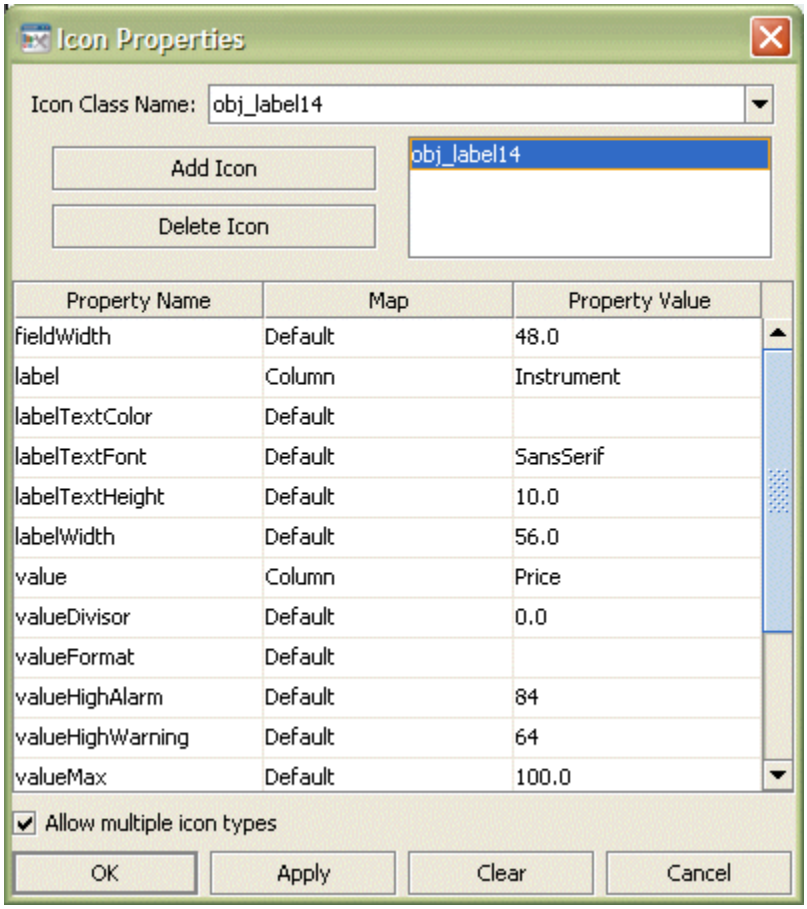
If you select Value, the value for the property is entered in the Property Value column. If you select Column, clicking in the Property Value column will display a list of all the columns in the tabular data.



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 add multiple objects for display in the grid.

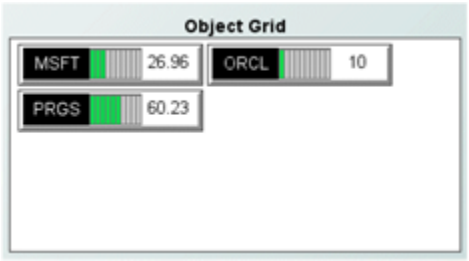


Use Add Icon and Delete Icon buttons to add and remove objects from the grid.

Using Object Grids

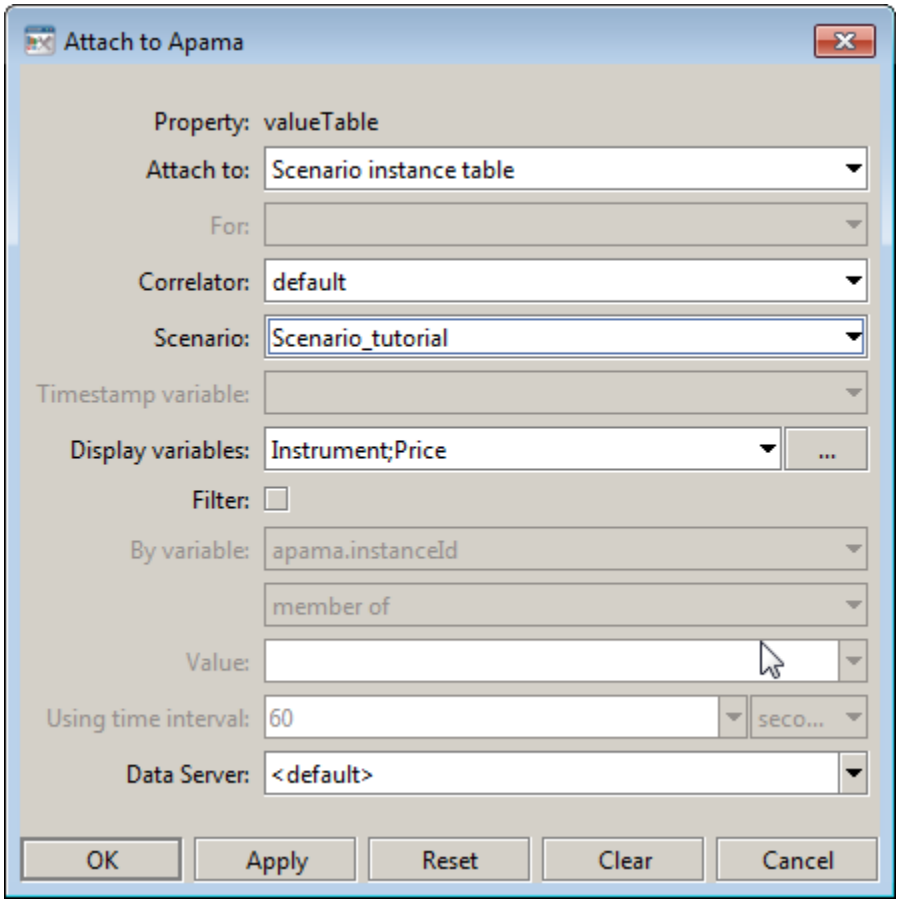
Recreating the Object Grid sample

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`.

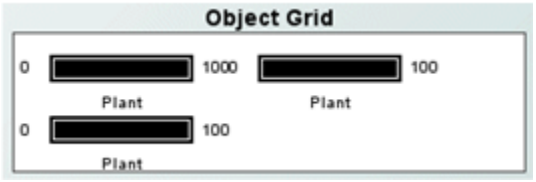


To recreate this sample, create a new dashboard and perform the following steps:

1. Add an Object Grid to the dashboard and attach its `valueTable` property to the tutorial scenario as follows.

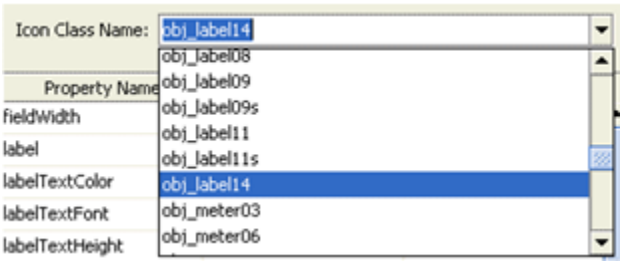


The grid object will update and display as follows:

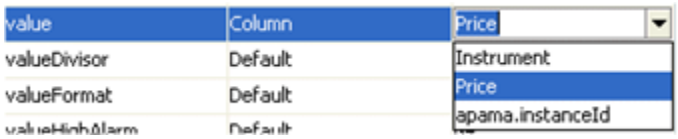


Unless you have separately created or deleted instances of the tutorial scenario, there will be three instances of the scenario and the grid will display three instances of the object. The objects do not show any values from the tutorial scenario because none of their properties have been bound to it in the **Icon Properties** dialog.

2. Select the grid object and double click on the `iconProperties` property to display the **Icon Properties** dialog. In the dialog select obj_label14 as the Icon Class Name.

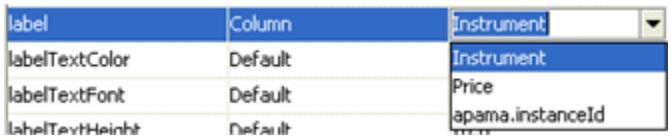


3. In the Icon Properties dialog click in the Map column of the `value` property and select the type Column. Click in the Property Value column and select Price.



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.



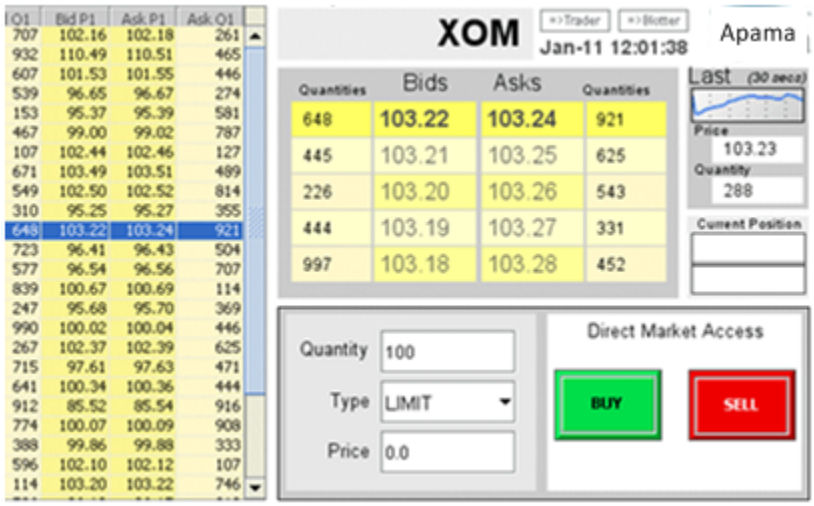
The dashboard should now appear similar to the Object Grid tutorial.

Using Object Grids

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:

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 155) is not supported inside of composite objects.

Reusing Dashboard Components

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.

Using Composite objects

Configuring Composite objects

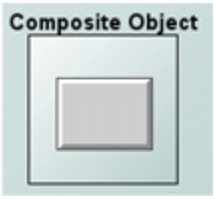
The Composite object is in the Composite tab of the object pallet.

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.

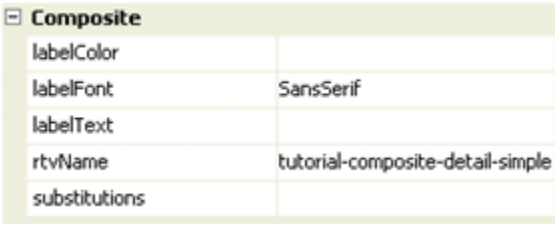
Note: This `rtv` file must not itself contain any composite objects. You cannot nest composite objects.

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.



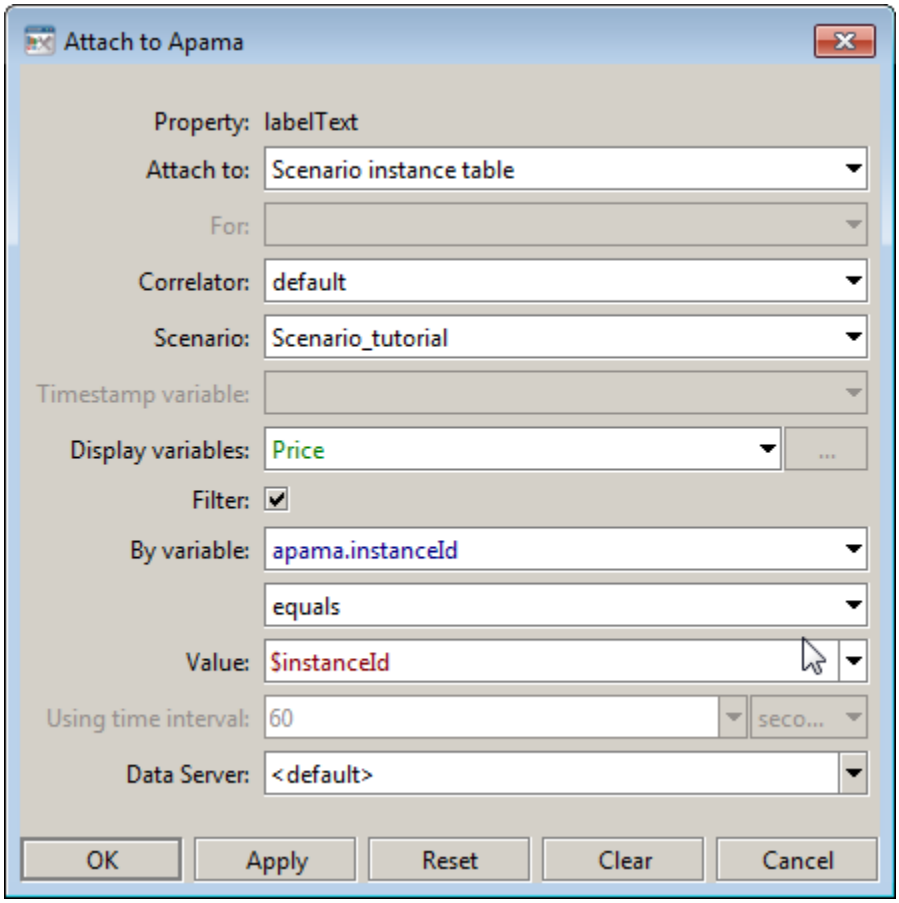
Here the labelColor, labelFont, and labelText variables are exposed as properties. Setting these you can change the appearance of the Composite object:

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.

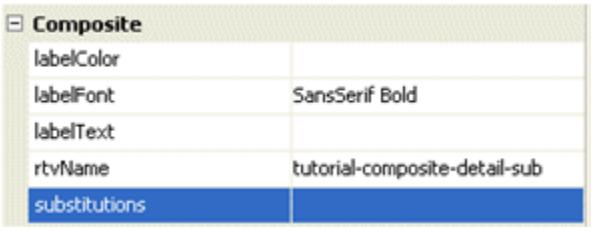


By attaching the property to the scenario and filtering by $instanceId, the `labelText` property will update when ever the value of the attachment changes. When the `labelText` property changes, it will change the value of the corresponding variable in the `rtv` file displayed in the composite which will in turn be reflected in the label.

Using Composite objects

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.

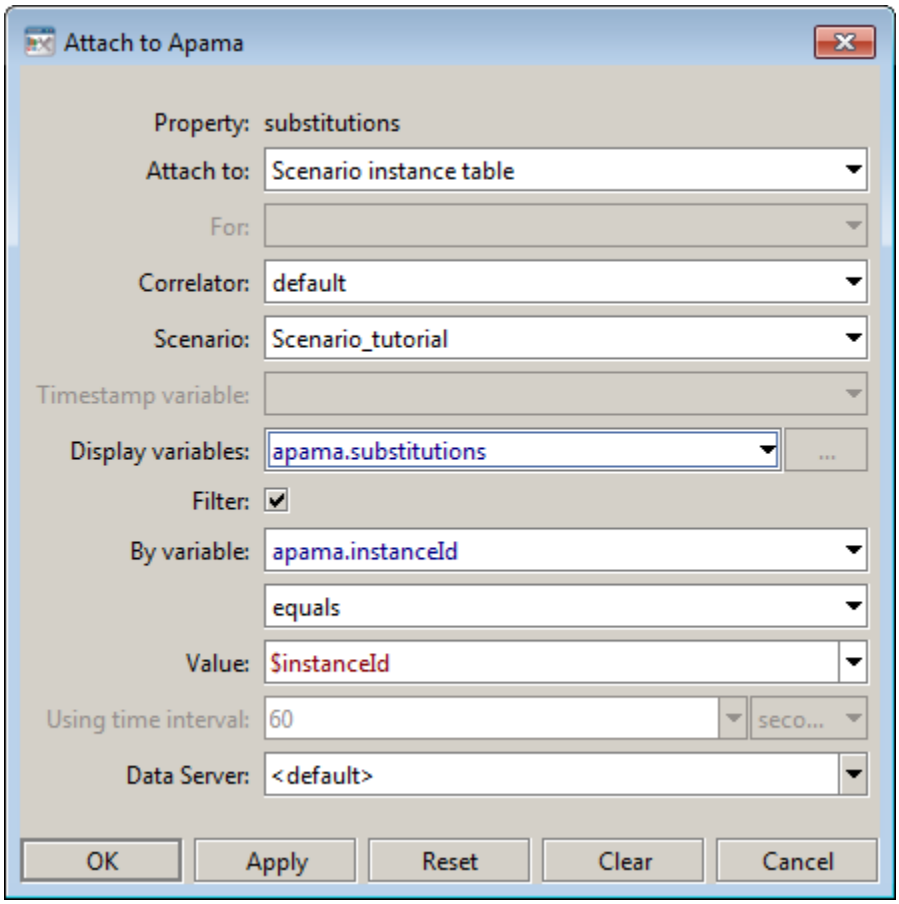
```
$subname:'Value with Spaces'
```

- Substitution names cannot contain the following characters:
  - :
  - |
  - .
  - tab
  - space
  - ,
  - ;
  - =
  - <
  - >
  - '
  - "
  - &
  - /
  - \
  - {
  - }
  - [
  -]
  - (
  -)

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.

Using Composite objects

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*.

Using Composite objects

Composite object sample

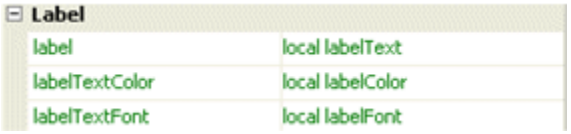
The Dashboard Builder tutorial includes an example of the Composite object.

- Open the file `tutorial-composite-simple.rtv` by selecting Composite Simple on the tutorial main page.

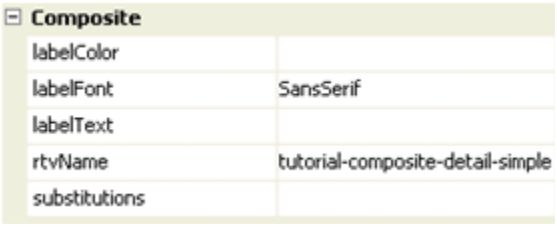
This dashboard displays a composite object with its `rtvName` property set to the file `tutorial-composite-detail-simple`.

- 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.



Using Composite objects

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.

Composite object sample

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.

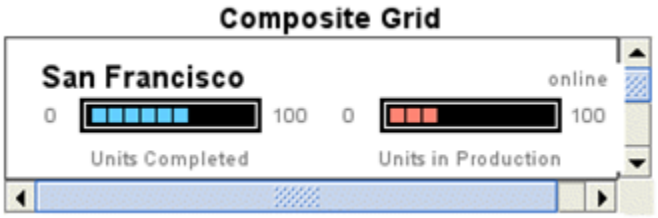
The objects are attached to the tutorial scenario filtering on $instanceId to select a single instance. The Composite Grid object is configured to pass each instance a unique value of $instanceId such that there is one row in the grid for each instance of the scenario.

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.

Reusing Dashboard Components

Configuring Composite Grids

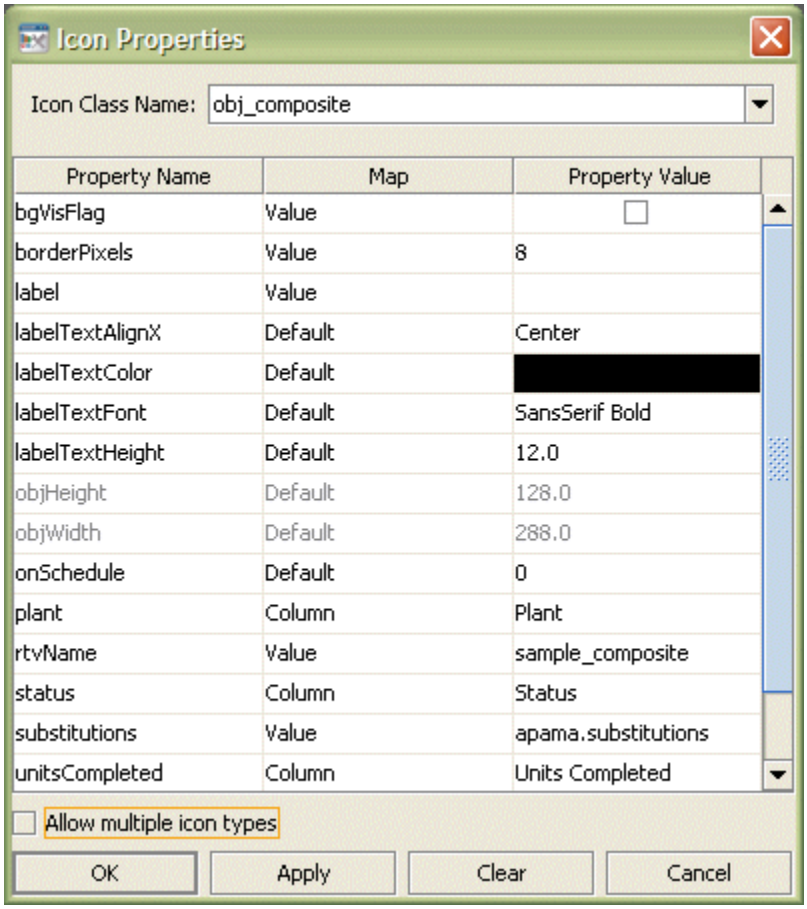
The Composite Grid is in the Composite tab of the object pallet.



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 on 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.

Using Composite Grids

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.

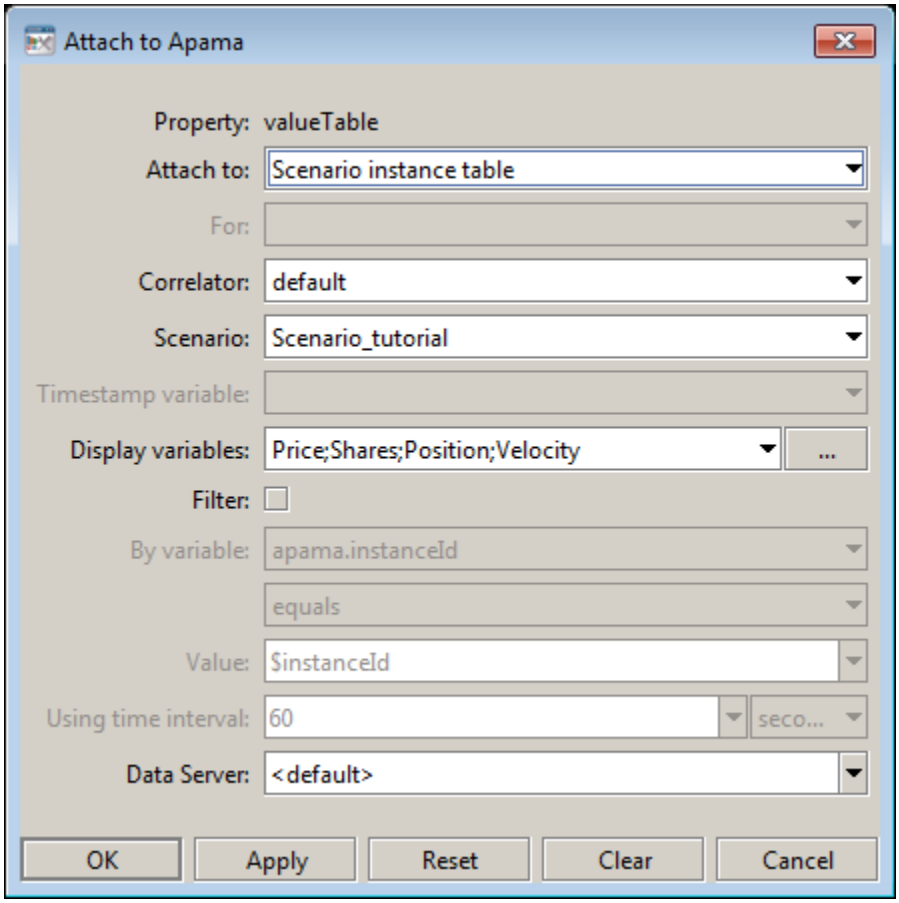
Examine the `label`, and `value` properties of the objects to see that they are attached to the tutorial scenario filtering on $instanceId. Range Dynamic objects are used to show Shares, Position, and Velocity. These objects are configured to change color to show if the value is positive, negative, or zero.

Using Composite Grids

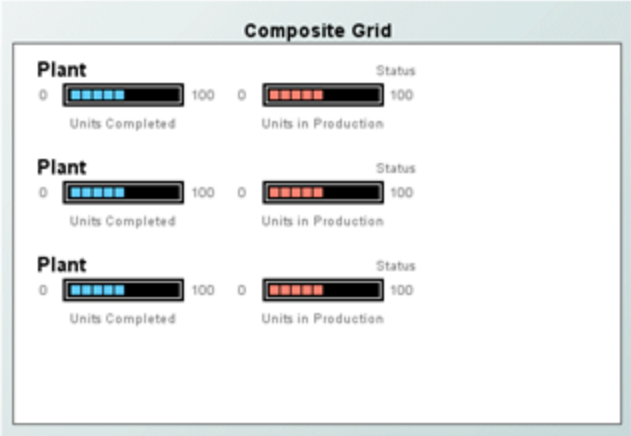
Recreating the Composite Grid sample

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:

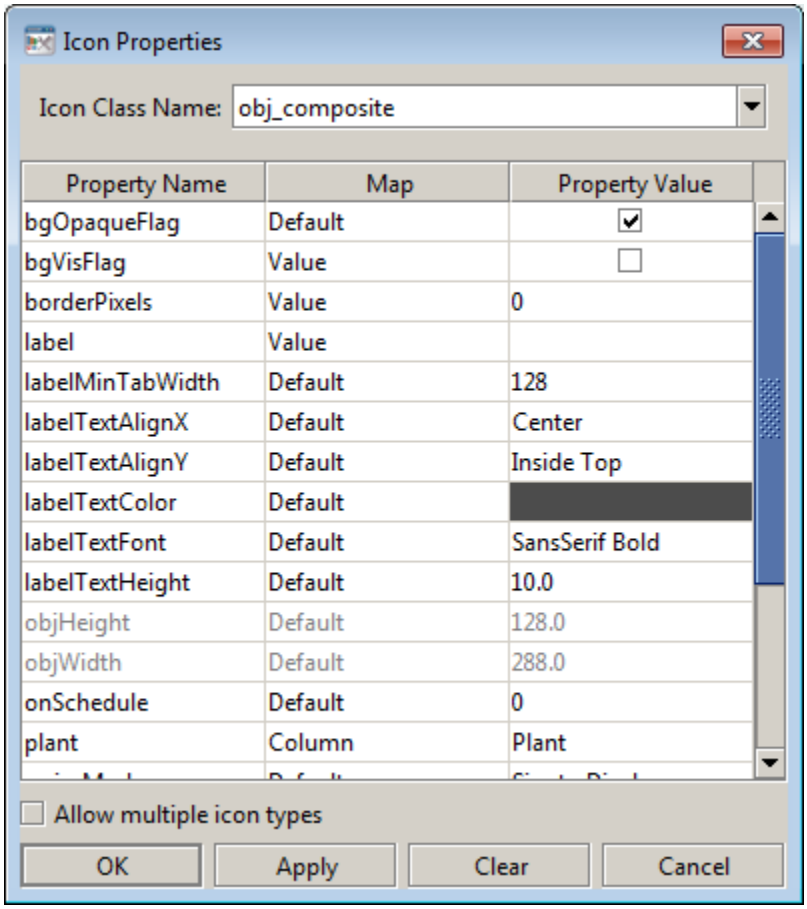


The Composite Grid will be similar to the following:



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 on 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.

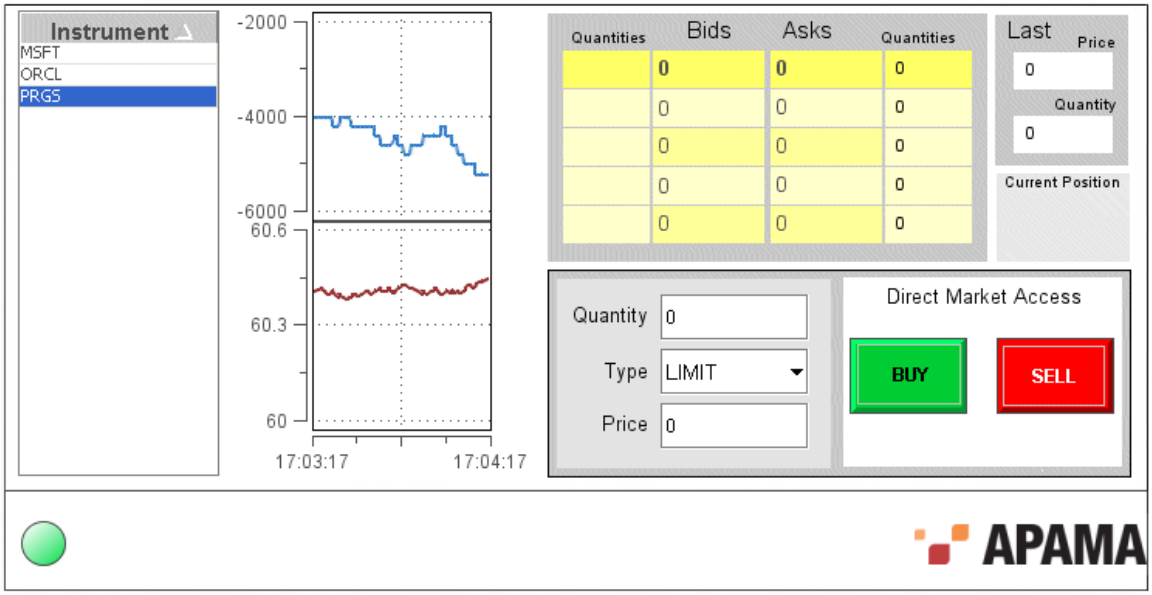
Composite Grid sample

Using include files

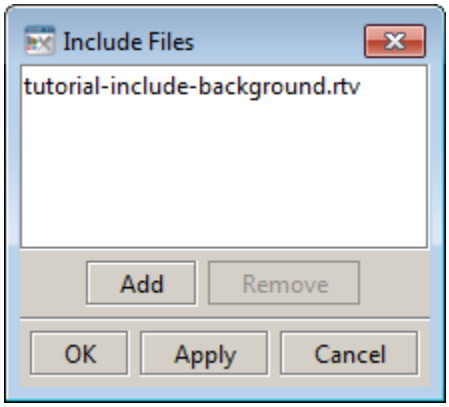
The Dashboard Builder include file feature provides a way to partition dashboard development and to reuse content in multiple dashboards. It allows you to include in a dashboard the objects, functions, and variables of another `rtv` file. Unlike the Composite object, the included file is not in an object; 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. The following illustration shows a simple status bar. Here the status bar contains an indicator (the green circle) of connectivity to the correlator, as well as the Apama logo. These objects could be defined in the `rtv` file `statusbar.rtv` file as follows:

The file `statusbar.rtv` could then be included in another dashboard.



To include an `rtv` file in a dashboard, select IncludeFiles 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.

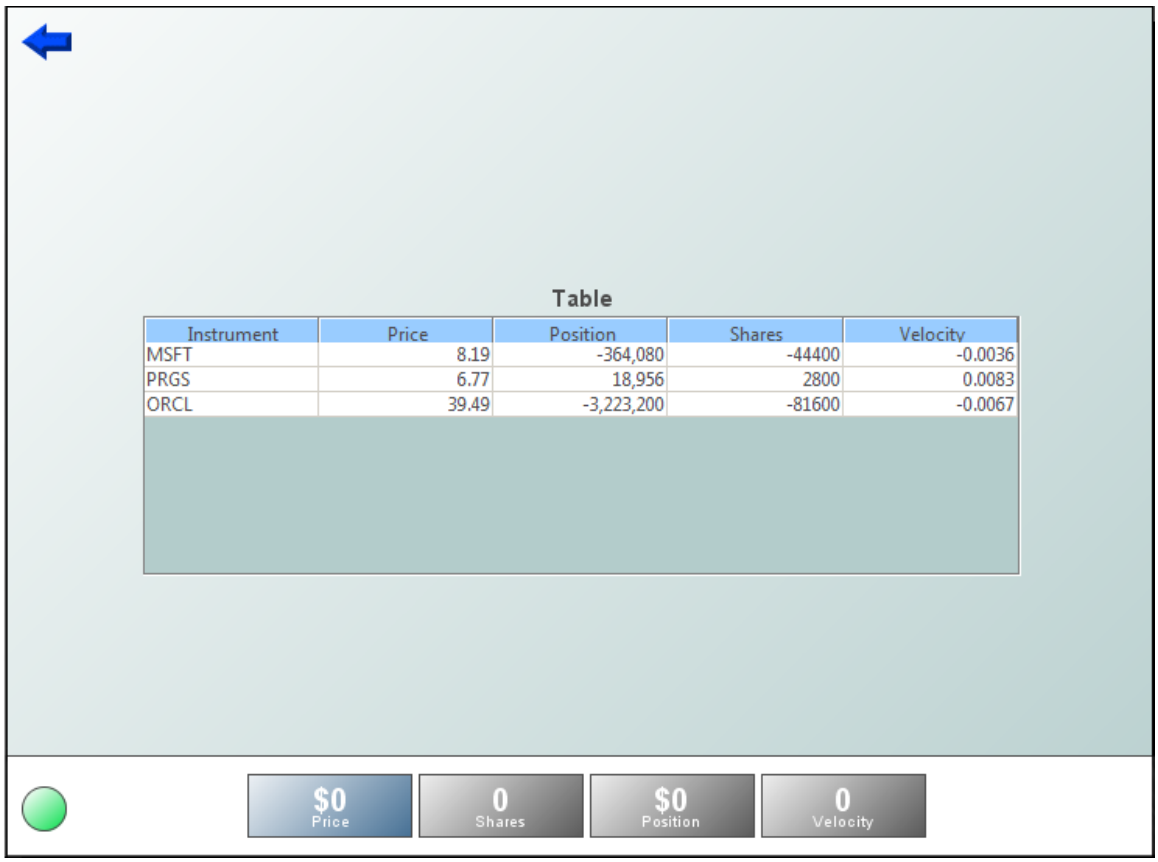
Substitutions such as $instanceId may be used in attachments in included files. Substitutions and variables in included files are scoped to the including dashboard. Runtime changes to their values will be reflected in included objects and functions. An attachment in an included file filtering on $instanceId will update whenever $instanceId changes in the dashboard.

Reusing Dashboard Components

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.

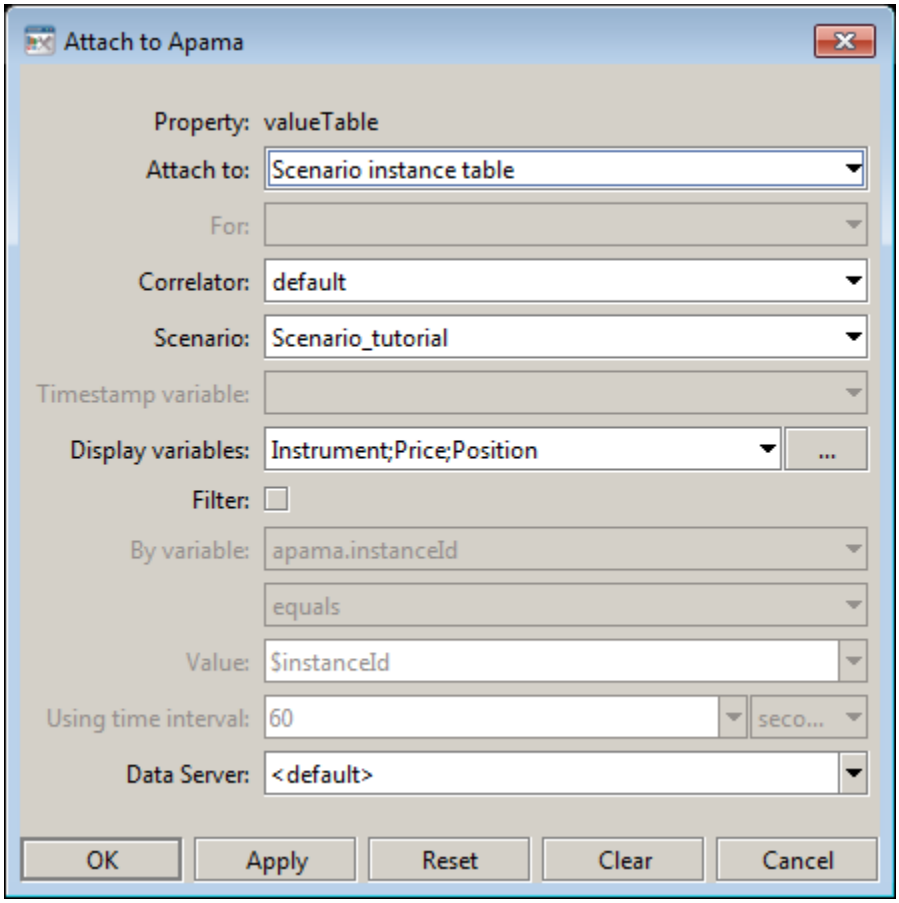
Examine the attachments of the `value` and `label` properties of the objects. Notice that they are attached to the tutorial scenario and filtering on $instanceId. No values are displayed because $instanceId does not have a value. It is set in the dashboard that includes this file.

Using include files

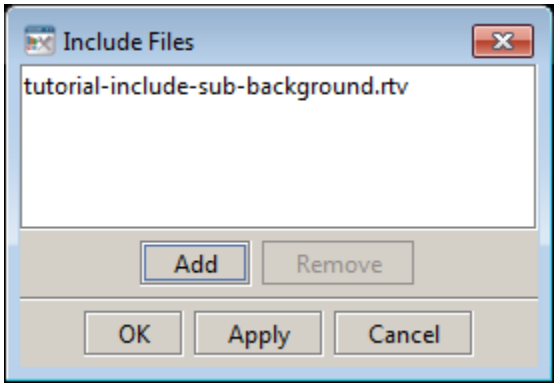
Recreating the Include File sample

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 on a row in the table to see values displayed in the included status bar.

Include File sample

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 in *Using Apama Studio*) or to the Dashboard Viewer executable (by using the `-c` or `--panelConfig` option—see the *Dashboard Viewer* guide).

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 diretory, the Display Server and Display Viewer look for it in the `lib` directory of your Apama installation directory.

Reusing Dashboard Components

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 188.

For information about the old tags, see "Using old tags to configure the panels in a window" on page 211.

Working with multiple display panels

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 draggable 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 substituions 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>
```

Working with multiple display panels

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.

Using new tags to configure the panels in a window

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 chnges 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 192.

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.

Using new tags to configure the panels in a window

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 191.
- `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 new tags to configure the panels in a window

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 , ; = <  > ' " & / \ { } [ ] ( )
  ```

Using new tags to configure the panels in a window

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 new tags to configure the panels in a window

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 190.
- 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 198.

Working with multiple display panels

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 execute 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 specifes the columns from the scenario instance table that will appear in the heirarchy 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 198.

Using tree controls in panel displays

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.

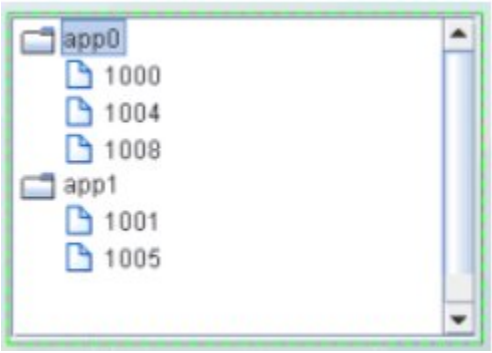
To illustrate how to create a tree using the row-leaf format, consider a table that has two columns, `App Name` and `PID`, and the following rows:

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`.



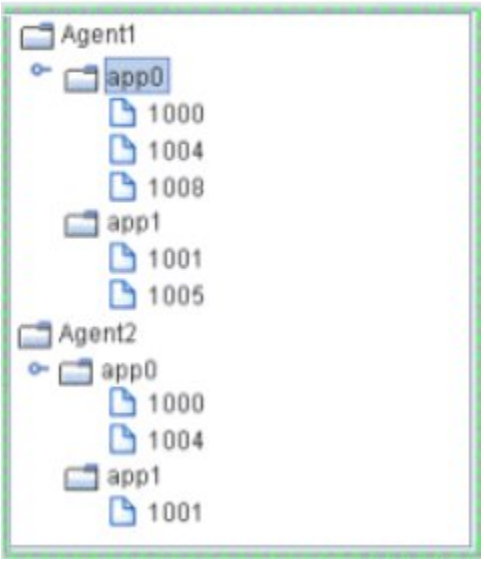
Suppose we add another column, `AgentName`, by selecting that variable from the Display Variables field of the Attach to Apama dialog. The table has the following rows:

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 198.

Creating tree controls

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 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

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 198.

Creating tree controls

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.

Using tree controls in panel displays

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 icons

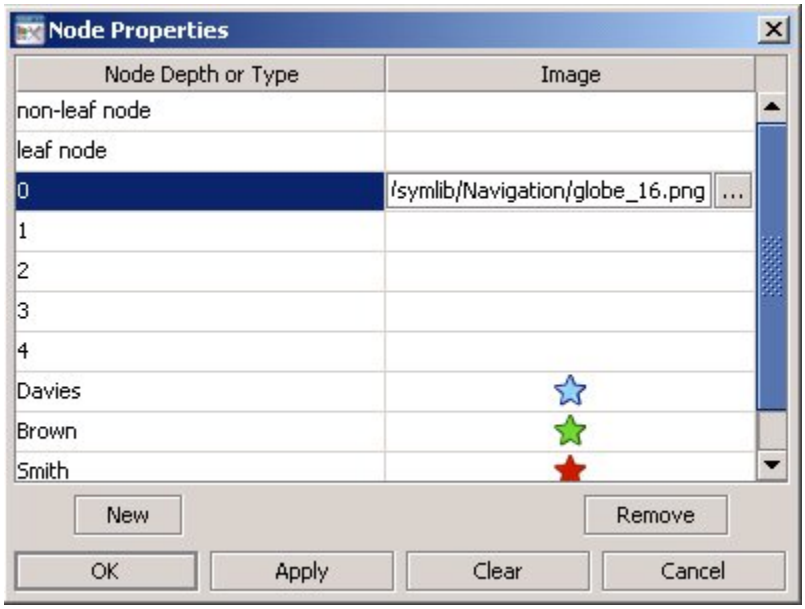
Configuring tree control type icons

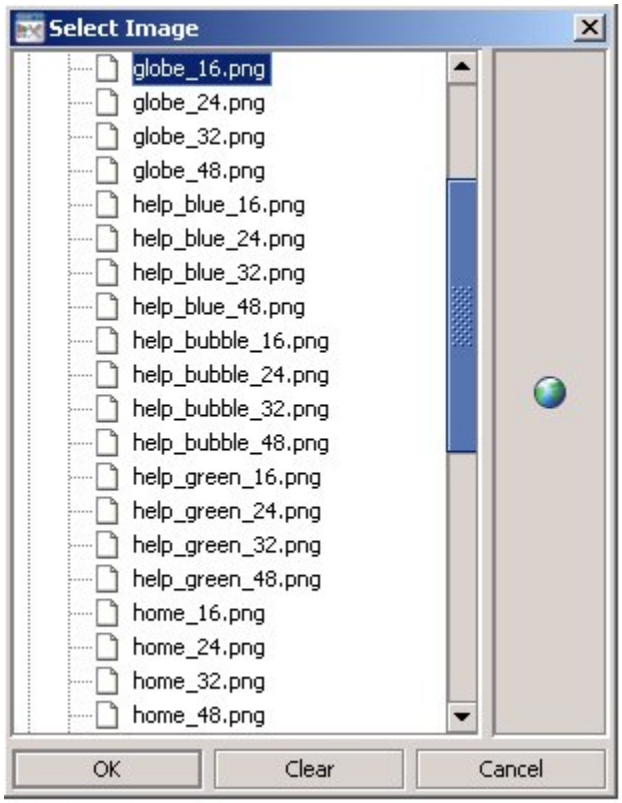
Type icons indicate the type of node in the tree. The type icon for each node is determined either by the value of a column in the `valueTable` property, or by the node position in the tree. By default, the type icon appears to the left of the node label.

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 icons

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 199.

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.

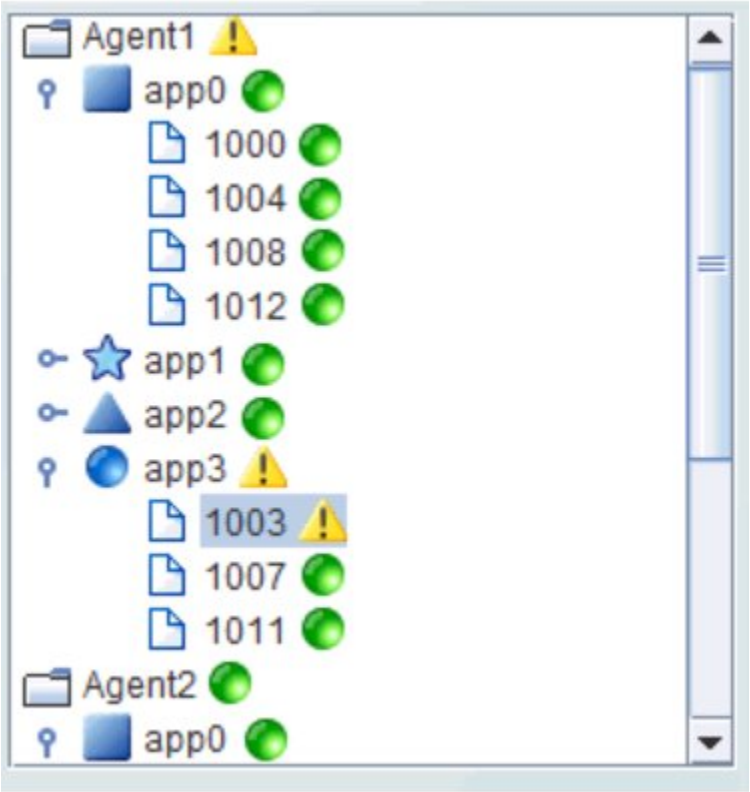
Click the  button in the Priority column for the node and assign an integer value: 0, 1, 2, 3, 4, 5, or a higher number. There is no upper limit on the number. The largest number is the highest priority and is propagated up the tree first. A value of zero (0) is not propagated. You might want to assign 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.



Configuring tree control icons

Specifying tree control properties

There are a number of properties that you can specify for a tree control.

Using tree controls in panel displays

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 properties

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 properties

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 properties

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 properties

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.

  NThe 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.

oThe 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 properties

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 displaythat 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.

Specifying tree control properties

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.

Specifying tree control properties

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 on 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 tree controls in panel displays

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. Your should change to the new tags. See "Using new tags to configure the panels in a window" on page 188.

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.JPanel`s with a `BorderLayout` Add a `JPanel` with a border layout to the main window. See "Using border panels" on page 212.
`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`s with a `CardLayout`. Display Server deployments do not support card panels. Add a

Tag	Description
	JPanel with a card layout to the main window. See "Using card panels" on page 213.
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 214.
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 215.
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 218.
RTViewPanel	Add a panel containing the specified display into a BorderPanel, CardPanel, TabbedPanel, or GridPanel. See "Using the RTViewPanel tag" on page 218.

Working with multiple display panels

Using border panels

The tags described in this topic are deprecated. They will be removed in a future release. Your should change to the new tags. See "Using new tags to configure the panels in a window" on page 188.

Use the BorderPanel tag to add a border panel to the main window. This tag supports the following attributes:

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 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 subelements 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. *Note:* 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
`long_panel.rtv`	south	962	120

Using old tags to configure the panels in a window

Using card panels

The tags described in this topic are deprecated. They will be removed in a future release. Your should change to the new tags. See "Using new tags to configure the panels in a window" on page 188.

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:

`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 old tags to configure the panels in a window

Using grid panels

The tags described in this topic are deprecated. They will be removed in a future release. Your should change to the new tags. See "Using new tags to configure the panels in a window" on page 188.

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:

`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 old tags to configure the panels in a window

Using tabs panels

The tags described in this topic are deprecated. They will be removed in a future release. Your should change to the new tags. See "Using new tags to configure the panels in a window" on page 188.

Use the `TabbedPanel` tag to arrange `.rtv` files into a tabbed panel. This tag supports the following attributes:

`placement`	Set the position of the tab. Valid arguments are `left`, `right`, `top`, and `bottom`. *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 old tags to configure the panels in a window

Using tree panels

The tags described in this topic are deprecated. They will be removed in a future release. Your should change to the new tags. See "Using new tags to configure the panels in a window" on page 188.

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:

`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:

`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: `label="<html><p style='font-family:monospaced;font-style:italic;font-size:50;color:green'>Your Label Goes Here"` 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:
	`$subname:subvalue $subname2:subvalue2`
	If a substitution value contains a single quote, it must be escaped using a `/` :
	`$filter:Plant=/'Dallas/'`
	If a substitution value contains a space, it must be enclosed in single quotes. Do not escape these single quotes:
	`$subname:subvalue $subname2:'sub value 2'`
	A substitution string cannot contain any of the following characters:
	`:` `|` `.` tab space `,` `=` `<` `>` `'` `"` `&` `/` `\` `{` `}` `[` `]` `(` `)`

The `treefont` tag supports the following attributes:

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:

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 old tags to configure the panels in a window

Using the RTViewNavTreePanel tag

The tags described in this topic are deprecated. They will be removed in a future release. Your should change to the new tags. See "Using new tags to configure the panels in a window" on page 188.

The `RTViewNavTreePanel` tag supports the following attributes:

`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 old tags to configure the panels in a window

Using the RTViewPanel tag

The tags described in this topic are deprecated. They will be removed in a future release. Your should change to the new tags. See "Using new tags to configure the panels in a window" on page 188.

The `RTViewPanel` tag supports the following attributes:

`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.
`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:
	`$subname:subvalue $subname2:subvalue2`
	If a substitution value contains a single quote, it must be escaped using a `/` :
	`$filter:Plant=/'Dallas/'`
	If a substitution value contains a space, it must be enclosed in single quotes. Do not escape these single quotes:
	`$subname:subvalue $subname2:'sub value 2'`
	A substitution string cannot contain any of the following characters:
	`: | .` tab space `, = < > ' " & / \ { } [ ] ( )`
	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`.

Using old tags to configure the panels in a window

Chapter 7: Sending Events to Correlators

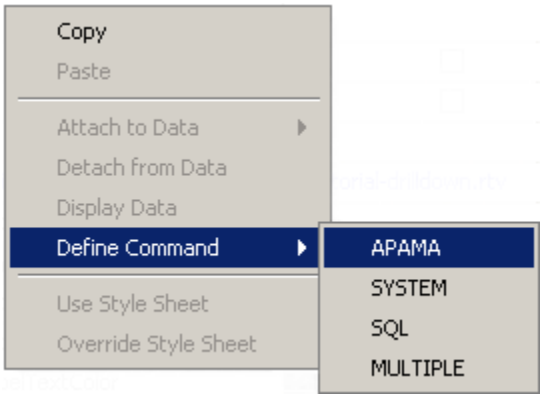
Dashboard Builder supports the creation of Dashboard commands that send user-defined events, just as it supports the creation of commands that send predefined Scenario manipulation commands.

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 on the `actionCommand` or `commandString` property in the Object Properties panel.
3. Select Define Command > Apama from the popup menu.

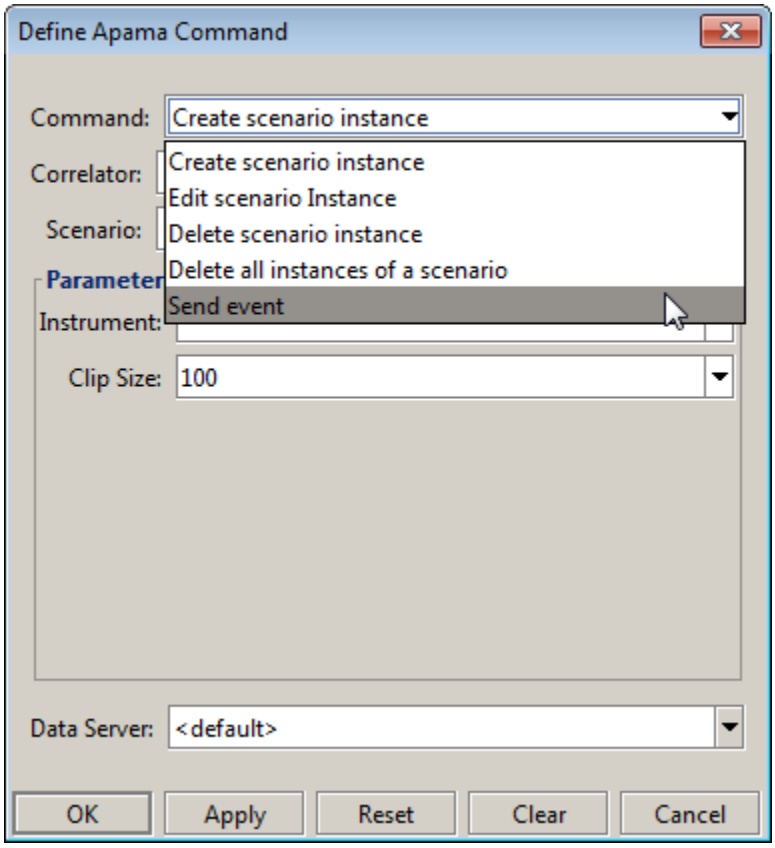


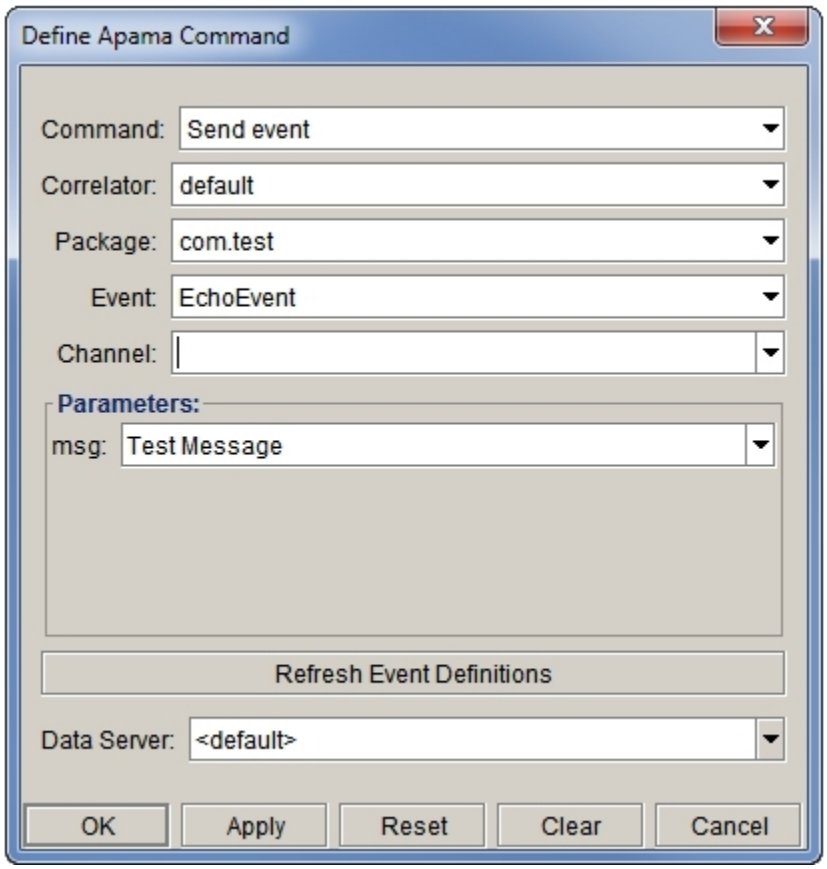
This displays the Define Apama Command dialog.

Sending Events to Correlators

Command field

The choices for the Command field include Send event.


Define Apama Command
Command:
Create scenario instance
Correlator:
Scenario:
Parameter
Instrument:
Clip Size:
100
Create scenario instance
Edit scenario Instance
Delete scenario instance
Delete all instances of a scenario
Send event
Data Server:
<default>
OK
Apply
Reset
Clear
Cancel


Define Apama Command
Command:
Send event
Correlator:
default
Package:
com.test
Event:
EchoEvent
Channel:
Parameters:
msg:
Test Message
Refresh Event Definitions
Data Server:
<default>
OK
Apply
Reset
Clear
Cancel

Using the Define Apama Command dialog

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".

Using the Define Apama Command dialog

Event field

The choices for the Event field include all the events for the selected package.

Using the Define Apama Command dialog

Channel field

Optionally, specify a channel on which to send the event. For example:

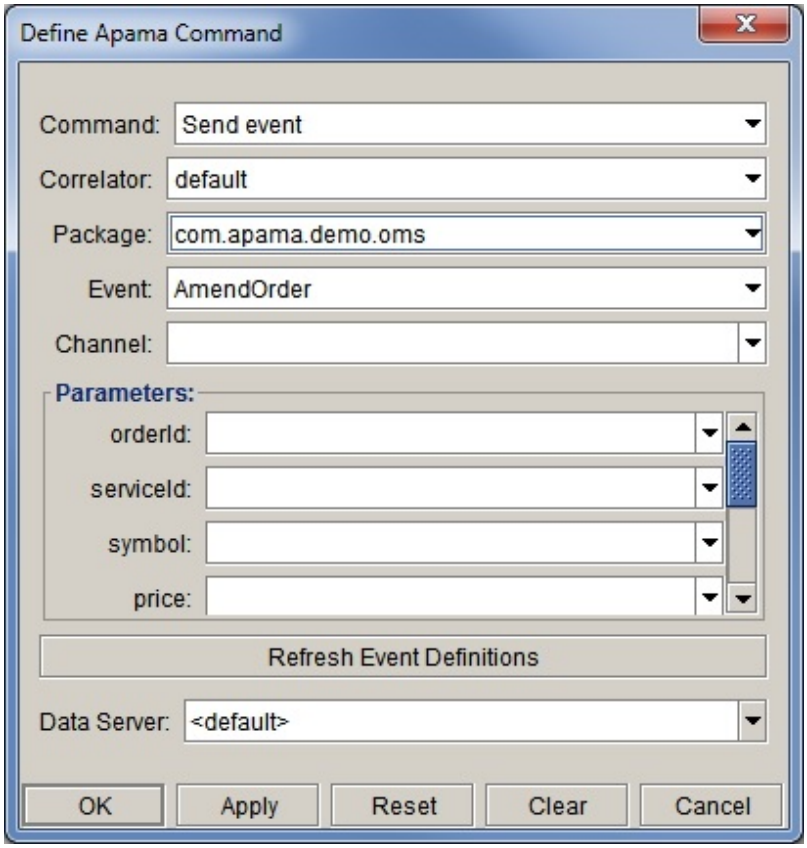
```
orders
```

If you do not specify a channel then the default channel is used.

Using the Define Apama Command dialog

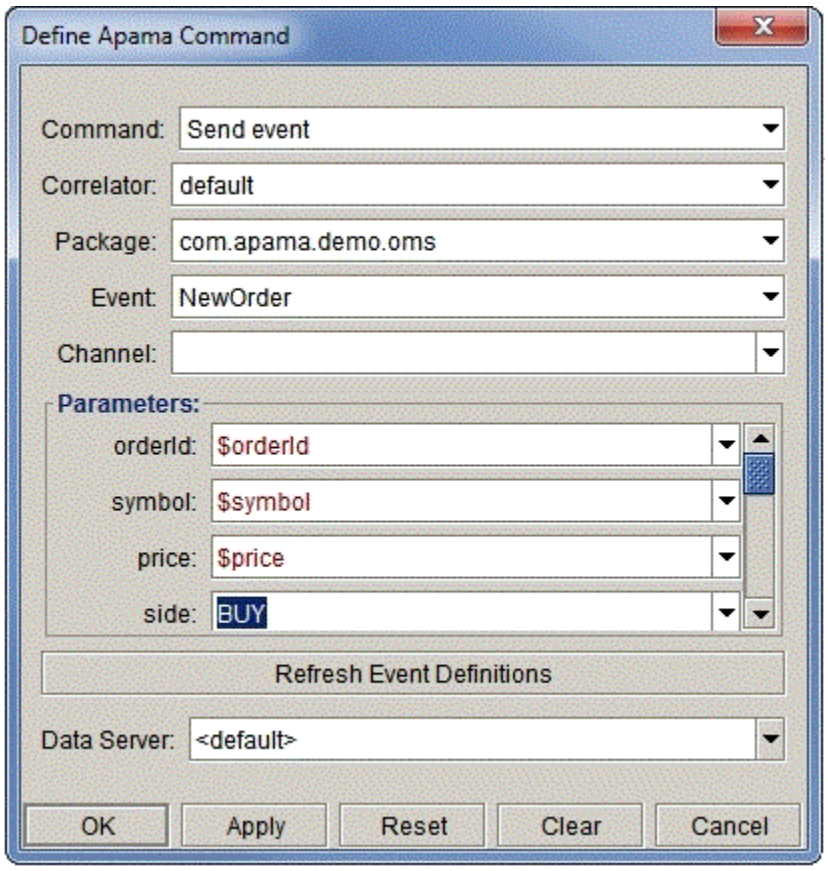
Other dialog fields

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:



Using the Define Apama Command dialog

Default values

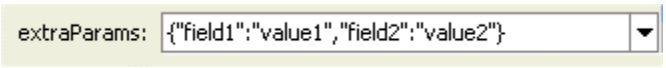
It is not necessary to set each event field in order to send an event. Empty fields are set to default values depending on type:.

Type	Default Value
`boolean`	`false`
`integer`	`0`
`float`	`0.0`
`string`	`""`
`location`	`(0.0,0.0,0.0,0.0)`
`sequence`	`[]`
`dictionary`	`{}`
`event`	*`event_name (default fields)`*

Other dialog fields

Specifying values for complex types

You specify values for complex types (`location`, `sequence`, `dictionary`, or `event`) by using the format specified in the *MonitorScript 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"}`.

Other dialog fields

Updating event definitions in Builder

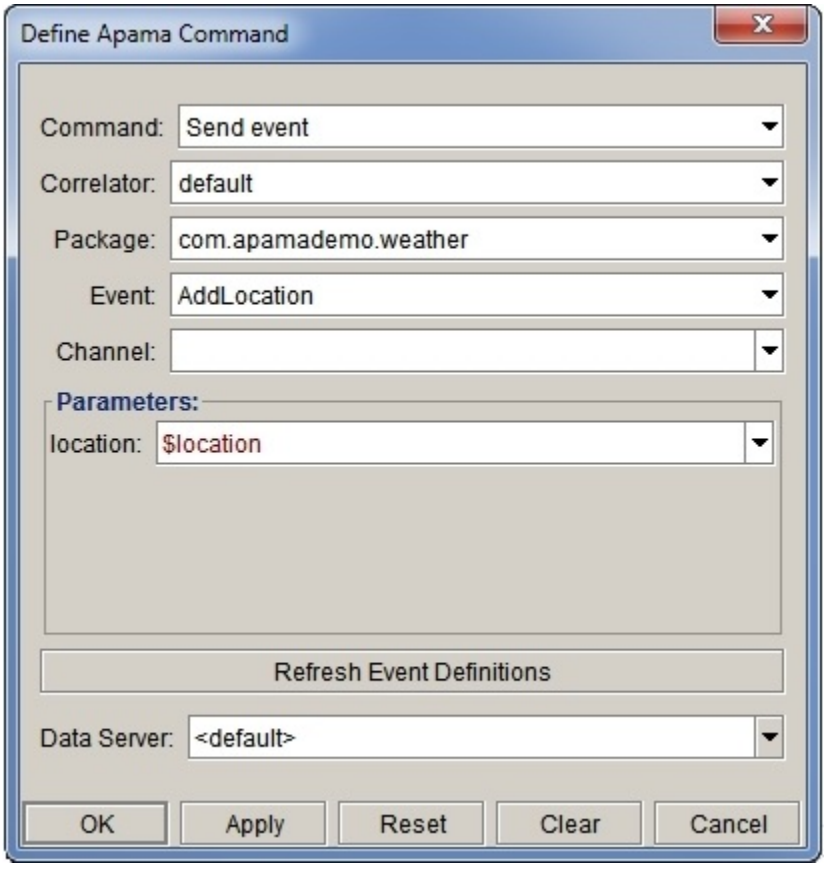
The Dashboard Builder retrieves the latest event definitions from each correlator at startup. If the event definitions change, you can force them to be refreshed by using the Refresh Event Definitions button at the bottom of the **Define Apama Command** dialog.

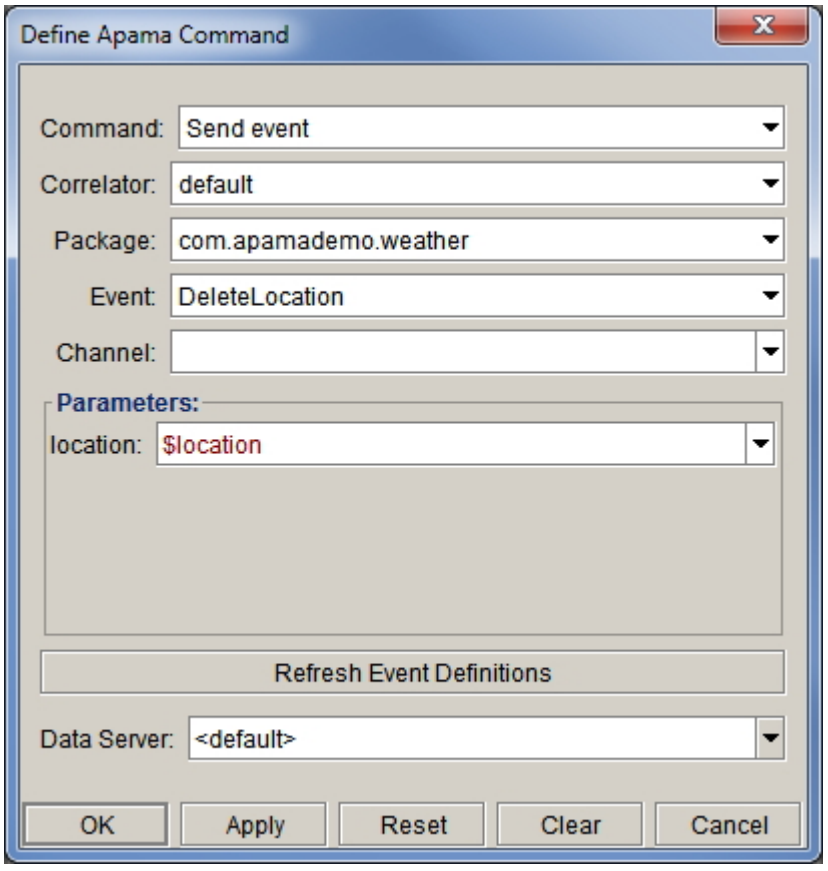
Note that the Refresh Event Definition button only updates the event definitions for the selected correlator.

Using the Define Apama Command dialog

Example

The Weather sample, available at the Apama Studio **Welcome** page, 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


Define Apama Command
Command: Send event
Correlator: default
Package: com.apamademo.weather
Event: DeleteLocation
Channel:
Parameters:
location: $location
Refresh Event Definitions
Data Server: <default>
OK
Apply
Reset
Clear
Cancel

Using the Define Apama Command dialog

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.

Sending Events to Correlators

Chapter 8: Using XML Data

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 following sections. XML data files can contain multiple scalar and tabular data elements.

Using XML Data

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.

XML data format

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>
```

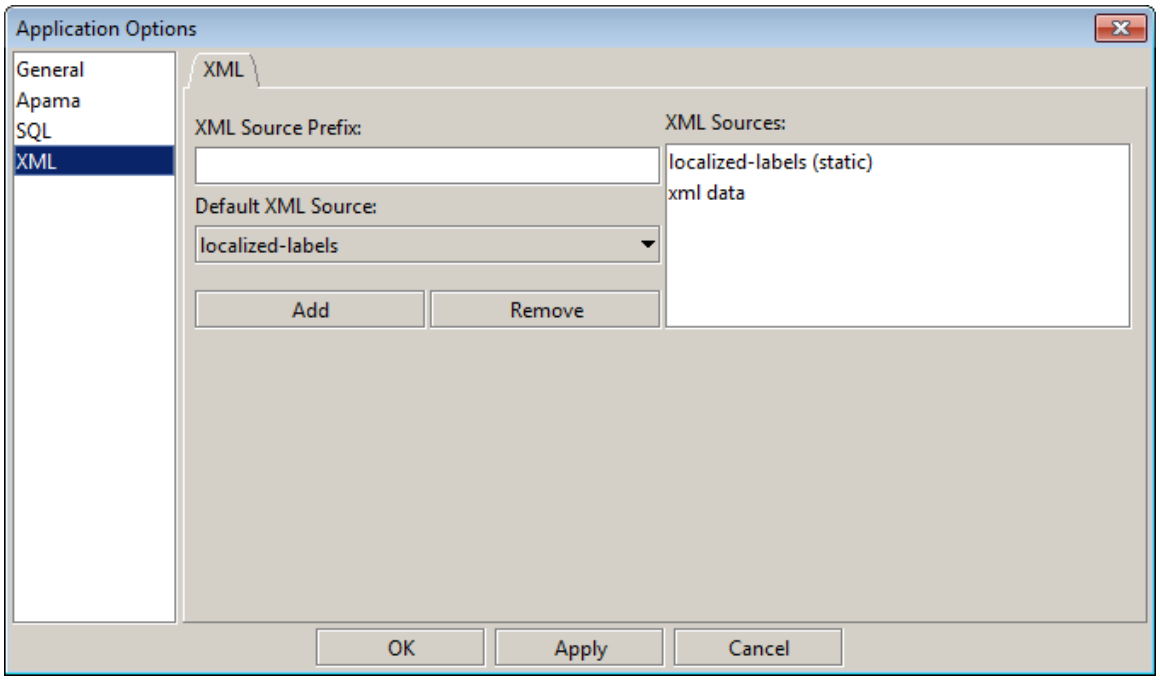
Here, the table is defined as containing four columns; `Plant`, `Units in Production`, `Units Completed`, and `On Schedule`. There are two rows in the table; one each for San Francisco and San Jose.

XML data format

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:

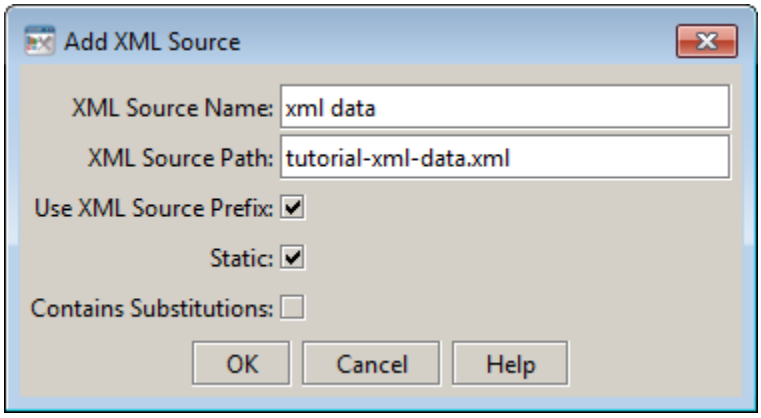
   ```
   %APAMA_HOME%\samples\dashboard_studio\tutorial\
   ```

   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 on 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 on 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.

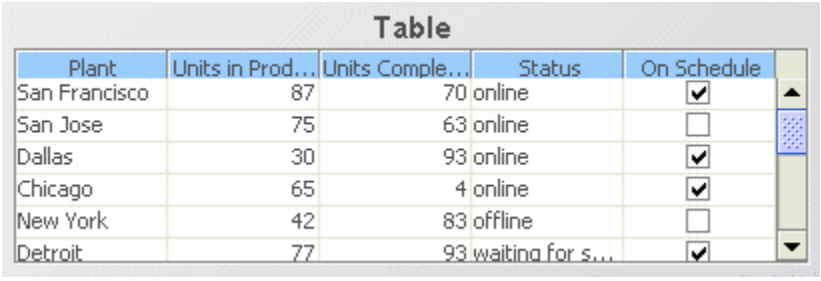
XML data source definitions are saved in `OPTIONS.ini`. To persist an XML data source definition you must click Save in the **Application Options** dialog.

Using XML Data

Attaching objects to XML data

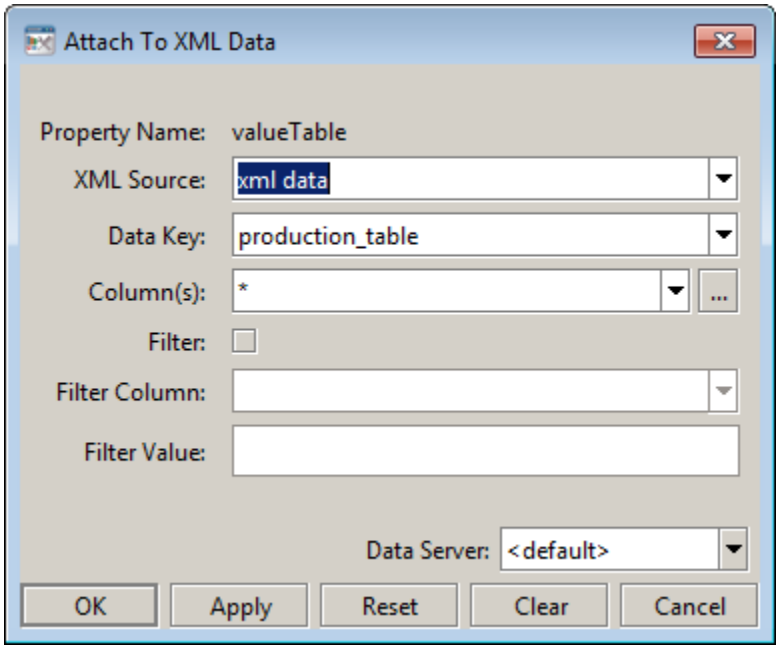
After having defined an XML data source you can attach object properties to the data elements within the XML data file. The steps for doing this are similar to defining attachments to scenario data.

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.



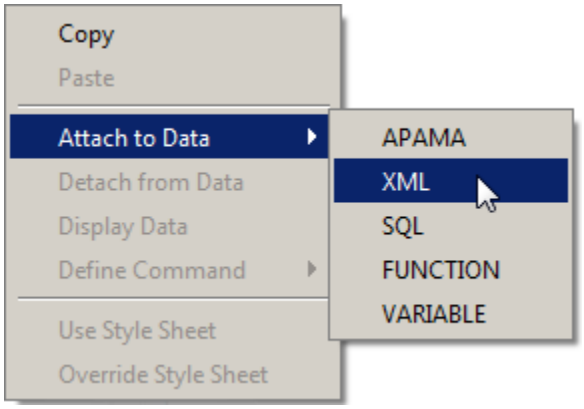
   The table object in this dashboard is attached to the `production_table` data element in the file `tutorial-xml-data.xml`.

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 on the `valueTable` property in the Object Properties panel.

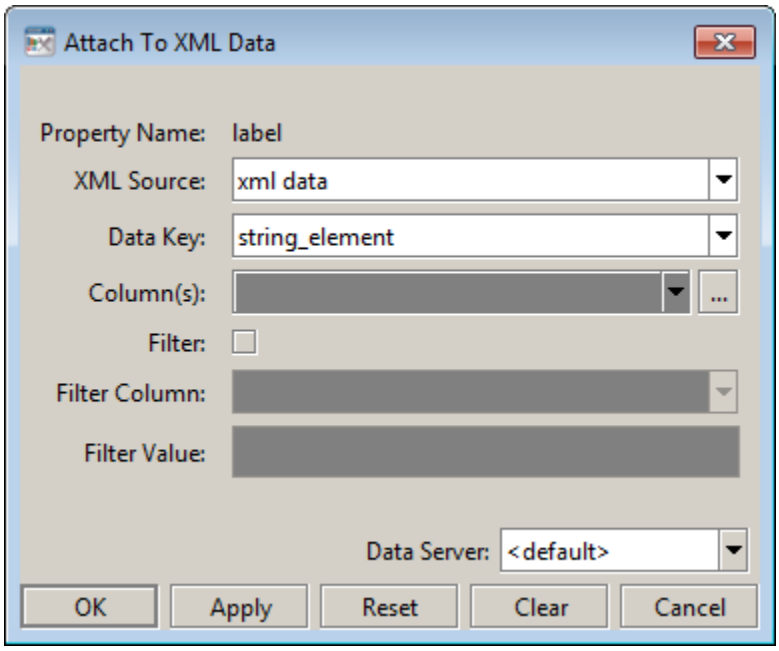


   Here the property is attached to the `production_table` data element for the XML data source named `xml data`. The `Columns` and `Filter` fields can be used to select a subset of columns or rows as is done for scenario data attachments.

5. With the table object still selected, right click the `label` property and select XML from the Attach to Data menu.

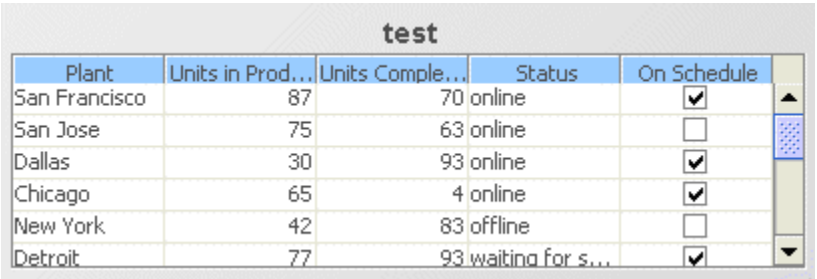


6. If `label` is a scalar property, it must be attached to a scalar data element. Attach it to the data element `string_element` as shown in the following:



Do not use the Data Server field of the **Attach to XML Data** dialog.

The label of the table will change.



Using XML Data

Chapter 9: Using SQL Data

The SQL data source provides access to JDBC or ODBC 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 or ODBC 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 246.

You must also modify your Dashboard Properties (select Properties from the Project menu in Apama Studio). In order to use a direct JDBC connection to communicate with a database, add your JDBC `jar` file to your Dashboard Properties.

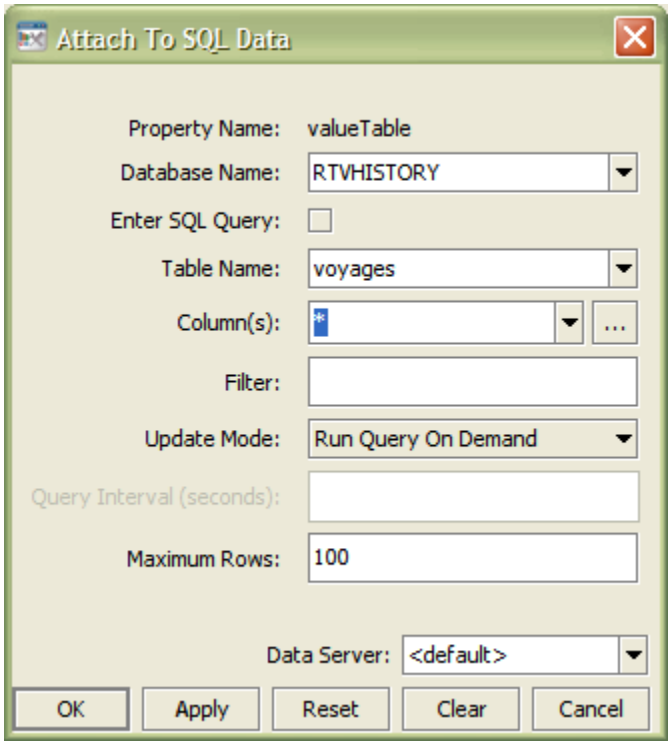
Using SQL Data

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 on 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 on 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.

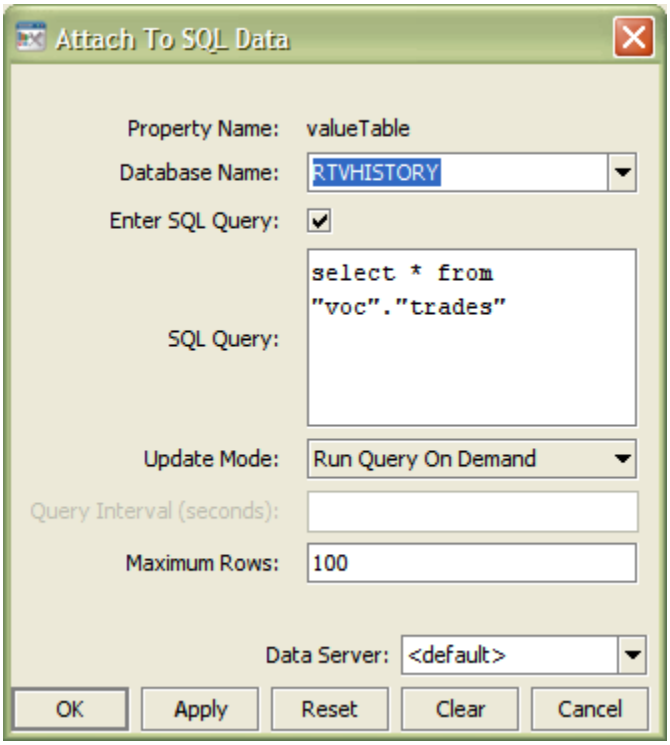
   The Database Name drop down menu lists all available databases. The Database Name field automatically displays the name of the default database. If the item you require is not listed, type your selection into the field.

   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 240 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.

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 246 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 240 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 240 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.

   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.

   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.

    By default Apama will attempt to communicate with your database using a JDBC-ODBC bridge connection that is not password protected. If you are using a direct JDBC connection or a password protected ODBC-JDBC bridge connection, you will need to add your database in Application Options | SQL.

Using SQL Data

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 240 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 an ODBC-JDBC bridge connection that is not password protected. If the validation response remains unknown, see "SQL tab" on page 241 for information on how to add a database. If you are using a direct JDBC connection or a password protected ODBC-JDBC bridge connection, you will need to add your database in Application Options.

Attaching visualization objects to SQL data

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.

Attaching visualization objects to SQL data

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 on 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 on 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 on the 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.

Attaching visualization objects to SQL data

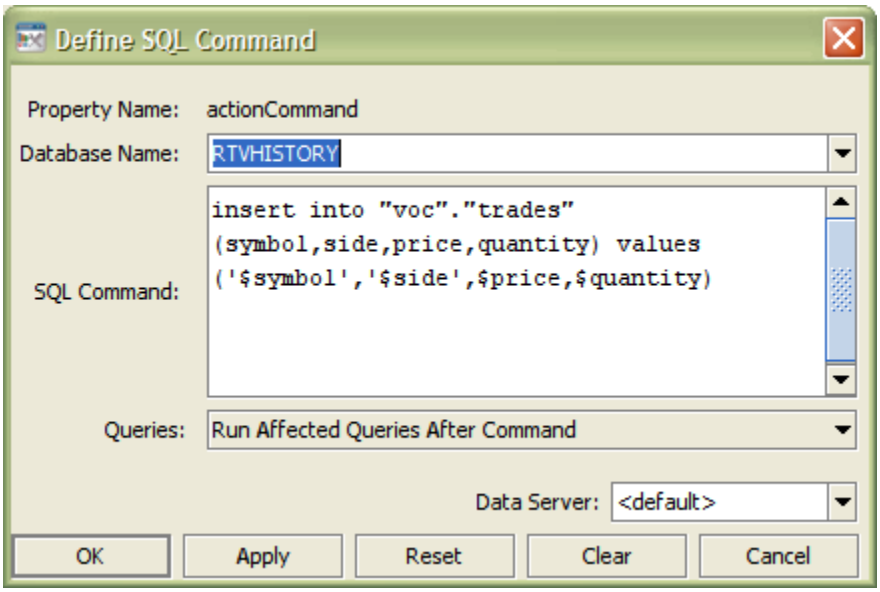
Defining SQL commands

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 on 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.

   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.

Using SQL Data

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 240 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 an ODBC-JDBC bridge connection that is not password protected. If the validation response remains unknown, see "SQL tab" on page 241 for information on how to add a database. If you are using a direct JDBC connection or a password protected ODBC-JDBC bridge connection, you will need to add your database in Application Options.

Defining SQL commands

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.

Defining SQL 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.

Using SQL Data

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 either a direct JDBC connection or an ODBC-JDBC bridge connection.

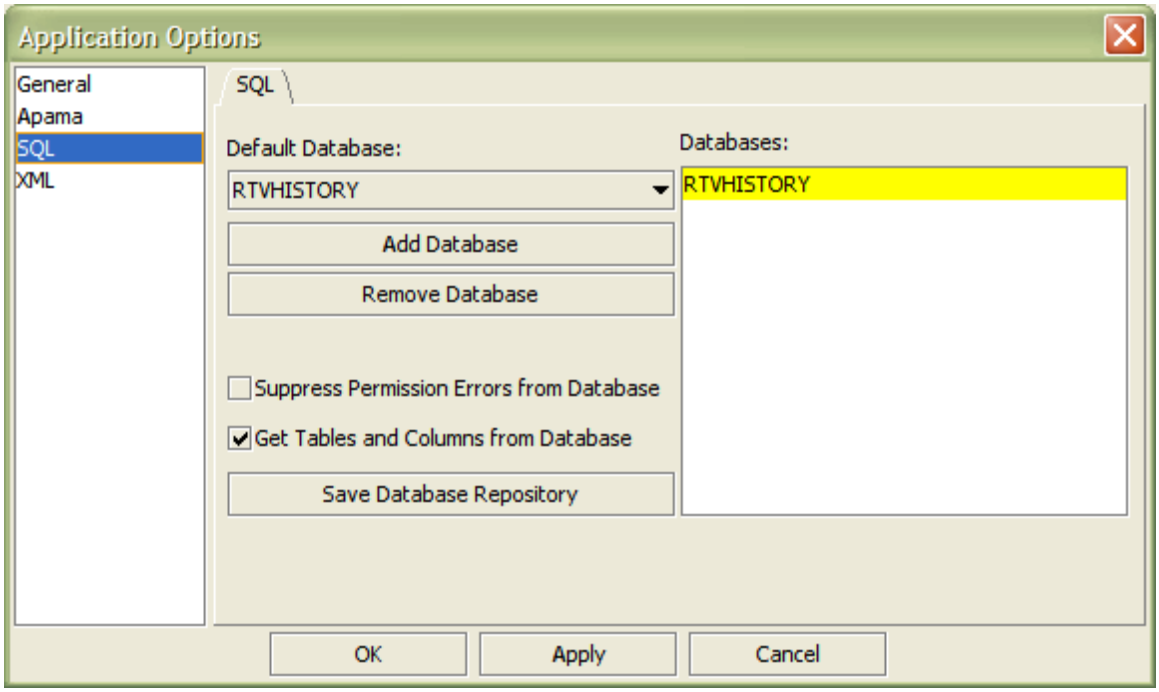
When you add a database to the list it will be highlighted in yellow indicating that it is not connected. To attempt to connect to a database, click OK, Apply, or Save. If the background remains yellow, then Apama was unable to make a connection to your database. *Note:* Databases that have been set 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 246 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.

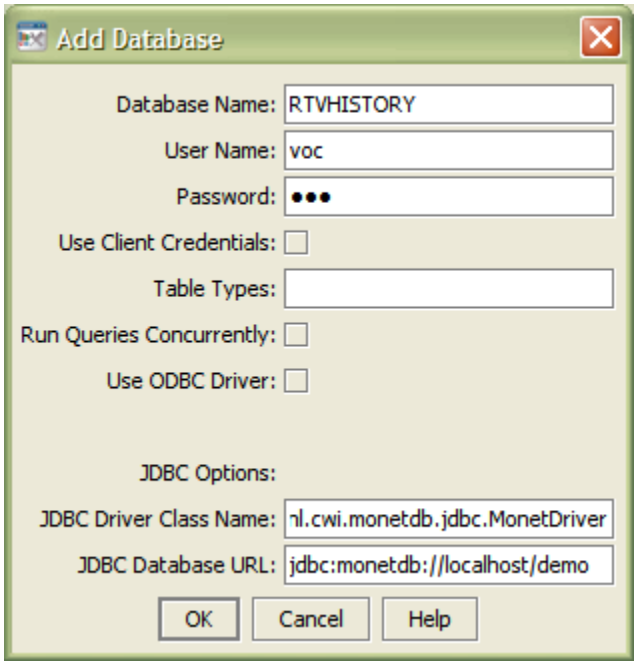
If you are using a direct JDBC connection or a password protected ODBC-JDBC bridge connection, you must click Save in order to record your options in `OPTIONS.ini`. This will allow Apama to reconnect with your database the next time you run the Builder or the Viewer.

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.



Specifying application options

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. If this connection will use an ODBC driver, this name must be the data source name used in the ODBC setup.
- 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.
- Use ODBC Driver: If selected, use an ODBC-JDBC bridge to connect to this database. An ODBC data source must be setup for this database to connect using an ODBC driver.
- 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 244.

Specifying application options

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 246 for details on editing an existing database repository file.

Specifying application options

Excluding tables From the Attach To SQL Data dialog

To exclude tables from the **Attach to SQL Data** dialog, see "Setting up SQL database connections" on page 246 for details.

Specifying application options

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 243), 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 ODBC databases, use "**-**" for `jdbcUrl` and `jdbcClassName`. For JDBC databases these fields must be set.

See also "Generating encrypted passwords for SQL data sources" on page 245.

Specifying application options

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 244), 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
```

Entering database information directory into OPTIONS.ini

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 an ODBC driver to connect to your database, modify your Java security settings to include the following permission:

```
permission java.lang.RuntimePermission "accessClassInPackage.sun.jdbc.odbc";
```

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.

Using SQL Data

Setting up SQL database connections

Apama communicates with your database using either a direct JDBC connection or an ODBC-JDBC bridge connection. Both connections require 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 241). 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 ODBC and JDBC database drivers for the following Apama-certified databases (note, the ODBC drivers are available only for Windows platforms):

- DB2
- Microsoft SQL Server
- Oracle

These database drivers eliminate the need to install database-vendor-supplied drivers. The ODBC database drivers are licensed to be used only with the Apama ADBC adapter. 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 locations:

apama_install_dir\doc\db_drivers\jdbc\books.pdf
apama_install_dir\doc\db_drivers\odbc\books.pdf

Using SQL Data

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 241, 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 Apama Studio).

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 Apama Studio).
2. Add the path to the JDBC driver `jar` file to the `APAMA_DASHBOARD_CLASSPATH` environment variable. 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 up SQL database connections

ODBC-JDBC bridge connection

In order for Apama to communicate with your database using an ODBC-JDBC bridge, you must have an ODBC driver for your database. Most databases that run on Microsoft Windows come standard with an ODBC driver. You must also register your database with ODBC before accessing it from Apama. The name specified for the ODBC data source name during the ODBC driver setup is the name you will use when accessing the database from Apama.

Apama includes several ODBC database drivers that eliminate the need to install database-vendor-supplied drivers. For more information, see "Setting up SQL database connections" on page 246.

Setting up SQL database connections

Registering your database with ODBC

To register your database with ODBC on Windows:

1. From the Windows Control Panel, double-click on the ODBC Data Sources icon. If this icon is not listed, double-click on the Administrative Tools icon and then double-click on the Data Sources (ODBC) icon.

   This will open the **ODBC** dialog.
2. In the **ODBC** dialog, click Add.
3. In the **Create New Data Source** window, select the driver for which you want to setup a data source.
4. Click Finish to display the **Setup** dialog.
5. Enter a Data Source Name. This is the name you will use in Apama when creating data attachments.
6. 4. Click the Select button and choose your database.
7. 5. Click OK in the **Select Database, Setup**, and **ODBC** dialogs.

Note: Standard UNIX systems do not provide an ODBC driver. On UNIX systems, it is currently unsupported to set up an ODBC driver to communicate with your database.

Setting up SQL database connections

Using a database repository file

A Database Repository file may be used to populate the initial values of drop down menus in the **Attach to Data** dialog. See Application Options for information on how to create a Database Repository file.

It is possible to edit an existing Database Repository file, however the file name `sqlrepository.xml` cannot be modified. If `sqlrepository.xml` is not found in the specified directory or your current working directory, Apama will look in the `lib` directory. If the Database Repository file is not found, drop down menus will remain empty until databases are added from the **Application Options** dialog or typed directly into the **Attach to SQL Data** dialog.

To edit an existing Database Repository file, supported tags and attributes are as follows:

- `sqlrepository` tag, `xmlns` attribute: Top level tag that includes the namespace attribute `xmlns`, which must be defined as `www.sl.com` (`xmlns="www.sl.com"`).

- `db` tag, `name` attribute: Database name
- `table` tag, `name` attribute: Table name
- `col` tag, `Possible` attribute: Column value

An example Database Repository file:

```
<?xml version="1.0"?>
<sqlrepository xmlns="www.sl.com" version="3.0">
        <db name="SampleDB">
                <table name="production_table">
                        <col>Plant</col>
                        <col>Units in Production</col>
                        <col>Units Completed</col>
                        <col>Status</col>
                        <col>On Schedule</col>
                </table>
                <table name="system_table">
                        <col>System</col>
                        <col>Status</col>
                        <col>%Free Space</col>
                        <col>CPU Usage</col>
                        <col>On Site</col>
                </table>
                <table name="trade_table">
                        <col>Customer</col>
                        <col>Symbol</col>
                        <col>Shares</col>
                        <col>Purchase Price</col>
                        <col>Current</col>
                        <col>High</col>
                        <col>Low</col>
                </table>
       </db>
</sqlrepository>
```

Setting up SQL database connections

Excluding tables From the Attach To SQL Data dialog

To exclude tables from the **Attach to SQL Data** dialog, copy the `sqlrepository.xml` file to a new sqlexcludedtables.xml file and remove the table references that you want to include in the Attach to SQL Data dialog drop down menus. For example:

```
<?xml version="1.0"?>
<sqlrepository xmlns="www.sl.com" version="3.0">
        <db name="SampleDB">
                <table name="production_table">
                        <col>Plant</col>
                        <col>Units in Production</col>
                        <col>Units Completed</col>
                        <col>Status</col>
                        <col>On Schedule</col>
                </table>
        </db>
</sqlrepository>
```

Save the `sqlexcludedtables.xml` file to the `lib` directory of your Apama installation. Table information stored in `sqlexcludedtables.xml` will now be excluded from the initial values of Table Name drop down menus in the **Attach to SQL Data** dialog.

Note: It is not necessary to create a Database Repository file in order to use `sqlexcludedtables.xml`. Apama will still use the `sqlexcludedtables.xml` file to exclude tables from the **Attach to SQL Data** dialog. If you have an `sqlexcludedtables.xml` file and you click the Save Database Repository button, the new `sqlrepository.xml` file will not contain any of the tables listed in your `sqlexcludedtables.xml` file.

To create your own `sqlexcludedtables.xml` file without creating a Database Repository File, supported tags and attributes are as follows:

- `sqlexcludedtables` tag, `xmlns` attribute: Top level tag that includes the namespace attribute `xmlns`, which must be defined as `www.sl.com` (`xmlns="www.sl.com"`).
- `db` tag, `name` attribute: Database name
- `table` tag, `name` attribute: Table name to exclude

Note: The file name `sqlexcludedtables.xml` cannot be modified.

Setting up SQL database connections

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> | db:<name> | 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.
- `db`: Name of SQL database. Only databases using ODBC drivers can be added on the command line
- `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.

Using SQL Data