Style Scope

Welcome to Style Scope!

Style Scope is a dashboard design and interactive analysis package that allows you to explore, analyze, monitor, report, and collaborate on business and operational data, all in real time. You will find Style Scope to be agile, robust, and easy to use.

The initial sections of this Guide help you connect to your data and get started with dashboards. Work through these sections to become comfortable using Style Scope:

1. *Connecting to Your Data*
2. *Using the Portal*
3. *Transforming Your Data*
4. *Creating a Dashboard*
5. *Interacting and Collaborating*
6. *Boosting Dashboard Performance*
7. *Managing the Server*

Style Scope is produced by InetSoft Technology, the leader in innovative and scalable Business Intelligence software. Please feel free to contact us with any questions!

- Phone: +1.888.216.2353 (US) / +1.732.424.0400 (International)
- Email: info@inetsoft.com
- Web: http://www.inetsoft.com/

Connecting to Your Data

To get started with Style Scope, you will first create a *connection* to your data. This connection is called a *datasource*. Style Scope allows you to create different kinds of data sources to suit the way your data is stored (relational database, spreadsheet, etc.)

- To create a data source for a spreadsheet or text file, proceed to *Connecting to Text or Excel Data*.
- To create a data source for a relational database, proceed to *Connecting to a Database*.

Because a datasource provides a persistent connection to your data, changes that you make to your data by updating, modifying, or deleting records are automatically reflected in all dashboards and reports that use the datasource.

If you do not wish to create a persistent datasource, you can still *import* data from a text file, CSV file, or Excel file directly into a Data Worksheet. For more information, see *Importing a Data File*.

Connecting to Text or Excel Data

To use a text file or Excel file as a data source, follow the steps below. In this example, you will connect to the sample Excel file *OrderDataText.xls*. (This Excel file contains the same data as the database in *Connecting to a Database*.)

1. From the Windows *Start* Menu, select the 'Style Studio' shortcut in the 'Style Scope' program group. This starts Style Studio, the desktop development tool.

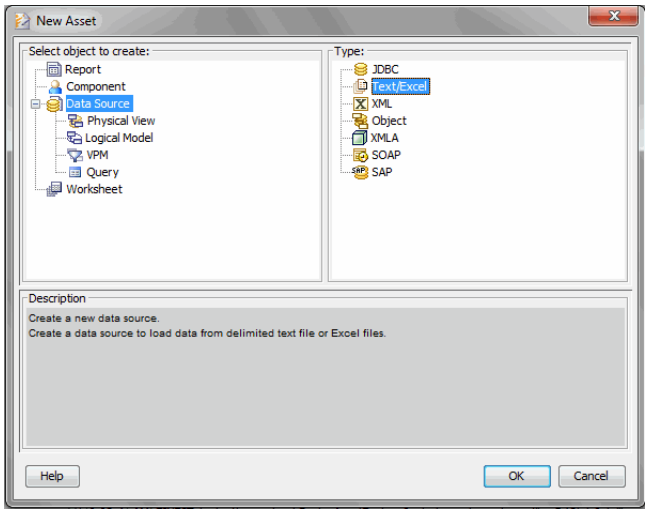


2. Press the 'New' button in the Style Studio toolbar. This opens the 'New Asset' dialog box.



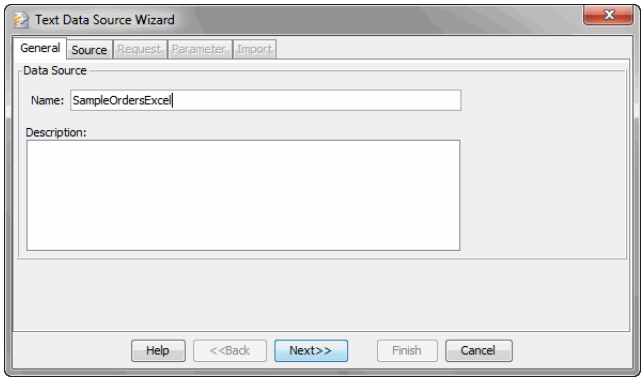
3. In the left panel of the 'New Asset' dialog box, press the 'Data Source' node on the tree.

4. In the right panel, select 'Text/Excel', and press 'OK'.



This opens the Text Data Source Wizard.

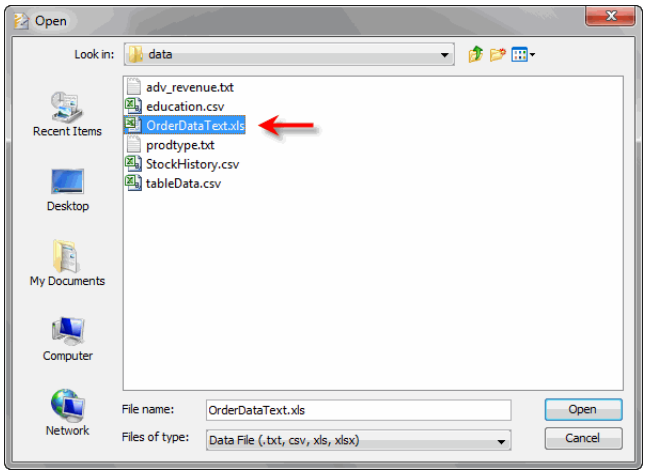
5. Enter "SampleOrdersExcel" in the 'Name' field. This is the name that will be used to identify the data source. Press 'Next'.



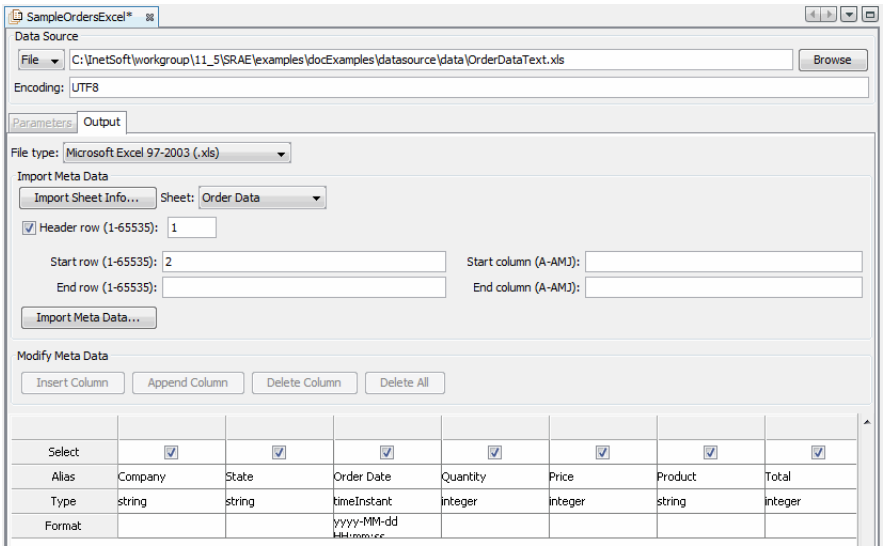
6. Press the 'Browse' button and navigate to the following directory in the Style Scope installation folder:

   `{Style Scope}\examples\docExamples\datasource\data`

7. Select the *OrderDataText.xls* file and press 'Open'.



8. Press 'Finish' to display the final Data Source definition page.



9. Press the ‘Save’ button in the toolbar to save the new data source. After you save the “SampleOrdersExcel” data source, it will appear in the left-side **Asset** panel under the ‘Data Source’ node.



10. Verify that you can connect to the Excel file. Press the ‘Test Data Source’ button in the top toolbar. A message will inform you if the connection was successful.



You have now established a connection to your data. To retrieve data from the file, continue to *Creating a Query with the Wizard*.

Connecting to a Database

This section illustrates the basic procedure for establishing a connection to a JDBC database. A JDBC (Java Database Connectivity) database is a database that provides a Java driver to facilitate connections from Java applications. The example below uses the sample Derby database included with the installation. For specific information about other databases, see the following sections in *Advanced Data Modeling*:

– *SQL Server*
– *SQL Server Express*
– *Oracle*
– *DB2*
– *MySQL*
– *Microsoft Access*

For information on connecting to non-relational databases, see *Connecting to Other Data Sources*.

To use a JDBC database as a data source, follow the steps below:

Note that drivers for ODBC data sources must support prepared statements in order to execute queries.

1. Add your database driver JAR files into the *sree/WEB-INF/lib* directory in the server installation. (Drivers for Oracle, SQL Server, and MySQL are included in the installation. For other databases, download the driver files from the database vendor website.)

2. From the Windows *Start* Menu, select the 'Style Studio' shortcut in the 'Style Scope' program group. This starts Style Studio, the desktop development tool.

   If you are using Windows 2008, run Style Studio as Administrator. To do this, right-click the executable file and select 'Run as administrator'.

See *Configuring the Style Studio Classpath* for more details.

3. Add your database driver JAR files into the Style Studio classpath. (Drivers for Oracle, SQL Server, and MySQL are included in the installation. For other databases, you can download the driver files from the database vendor website.)

   To do this, follow the steps below:

   a. Select 'Configure' from the Style Studio 'File' menu. This opens the 'Style Studio Configuration' dialog box.

   b. Select the **Classpath** tab.

   c. Press 'Add JAR' to add a driver JAR file (e.g., a JDBC driver) to the classpath.

d. Press ‘OK’ when you have added all required drivers.



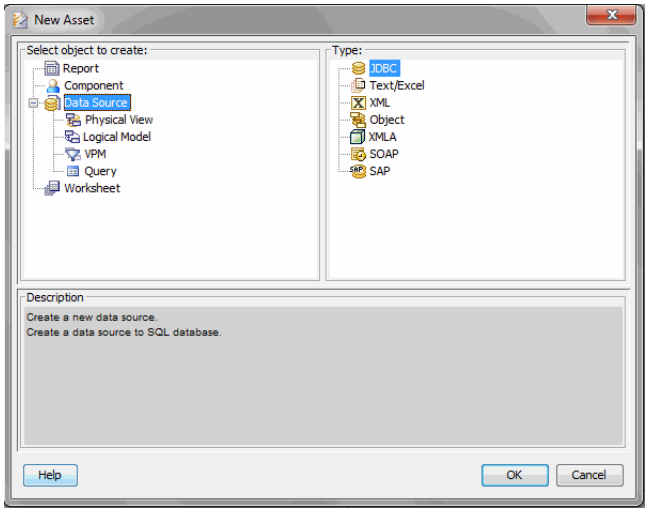
4. Press the ‘New’ button in the Style Studio toolbar. This opens the ‘New Asset’ dialog box.



5. In the left panel of the ‘New Asset’ dialog box, press the ‘Data Source’ node on the tree.

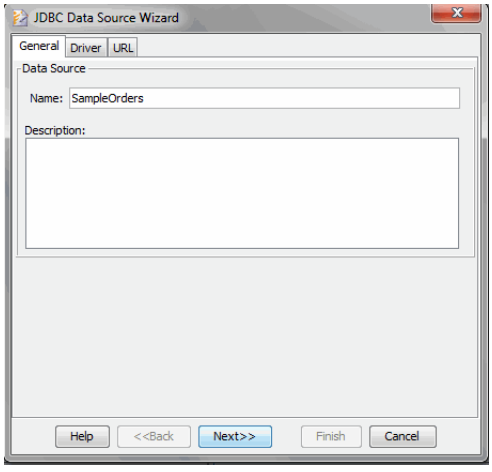
Most relational databases (Oracle, SQLServer, etc.) are “JDBC” databases, which means they provide a Java driver that allows Style Scope to connect to the database.

6. In the right panel, select ‘JDBC’, and press ‘OK’.

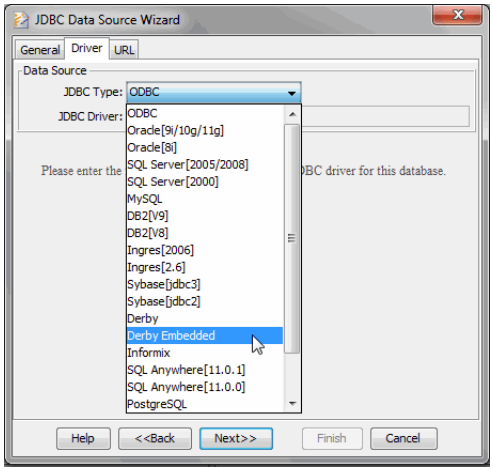


This opens the JDBC Data Source Wizard.

7. Enter “SampleOrders” in the ‘Name’ field. This is the name that will be used to identify the data source. Press ‘Next’.



8. Select ‘Derby Embedded’ from the menu for the ‘JDBC Type’. The correct driver, *org.apache.derby.jdbc.EmbeddedDriver*, automatically appears in the ‘JDBC Driver’ field.



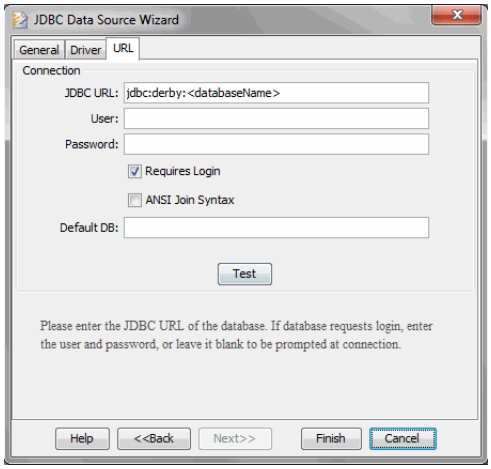
9. Press ‘Next’ to continue to the ‘JDBC URL’ page.

10. In the ‘JDBC URL’ field, enter the following:

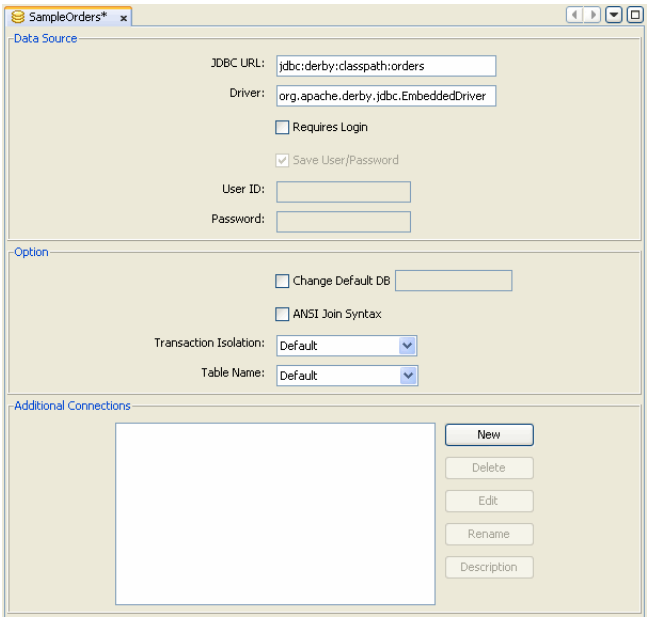
    ```
    jdbc:derby:classpath:orders
    ```

    This is the connection URL for the sample Derby database.

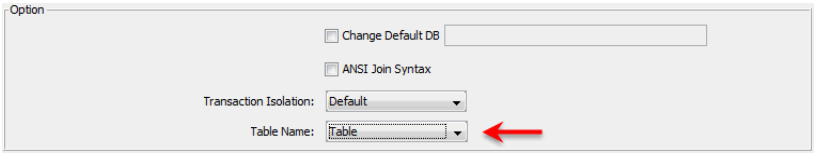
11. Uncheck the ‘Requires Login’ option.



12. Press ‘Finish’ to display the final Data Source definition page.



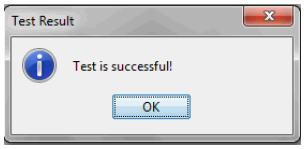
13. If you are configuring an ODBC datasource such as Microsoft Access, set the ‘Table Name’ option to ‘Table’.



14. Press the ‘Save’ button in the toolbar to save the new data source. After you save the “SampleOrders” data source, it will appear in the left-side **Asset** panel under the ‘Data Source’ node.



15. Verify that you can connect to the database. Press the ‘Test Data Source’ button in the top toolbar. A message will inform you if the connection was successful.



You have now established a connection to your data. To retrieve data from the database, continue to *Creating a Query with the Wizard*.

Style Studio caches meta-data for data sources that you configure. If you make a change to the database, such as adding or removing columns, To refresh the cached meta-data of your schema, open the datasource in Style Studio and press the ‘Refresh Metadata’ button in the toolbar.

Connecting to Other Data Sources

For instructions on how to connect to a typical JDBC database, see *Connecting to a Database*.

For information on how to connect to non-JDBC data sources, see the following sections in *Advanced Data Modeling*:

– *Generic Multidimensional Data Source*
– *SQL Server Analysis Services*
– *XML Data Source*
– *Web Service Data Source*
– *Text Data Source*
– *Java Object Data Source*
– *SAP Data Source*
– *Google Analytics Data Source*
– *SharePoint Data Source*

Importing a Data File

If you do not wish to create a persistent data source as described in *Connecting to Text or Excel Data*, you can still import a plain text file, CSV file, or Excel file into a Data Worksheet. In this example, you will import an Excel file (the same file used in *Connecting to Text or Excel Data*).

Follow the steps below to import an Excel file into a Data Worksheet.

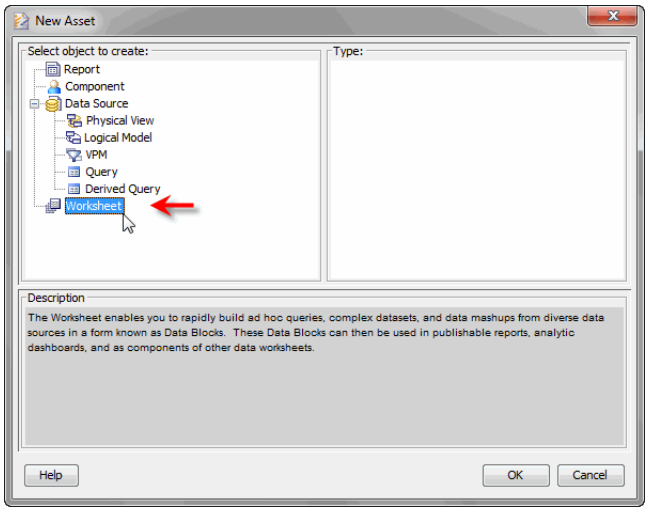
1. From the Windows *Start* Menu, select the 'Style Studio' shortcut in the 'Style Scope' program group. This starts Style Studio, the desktop development tool.



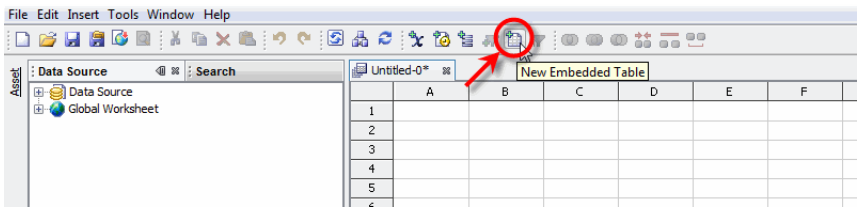
2. Press the 'New' button in the Style Studio toolbar. This opens the 'New Asset' dialog box.

Users can also create a new Worksheet in the User Portal. See *Creating a Worksheet in the User Portal* in *Transforming Your Data*.

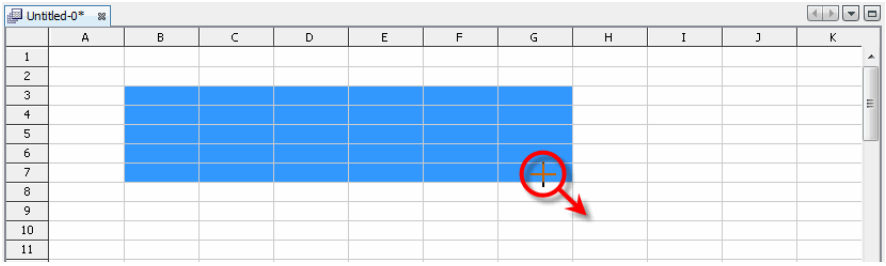
3. In the left panel, select 'Worksheet', and press 'OK'. This opens a new Data Worksheet.



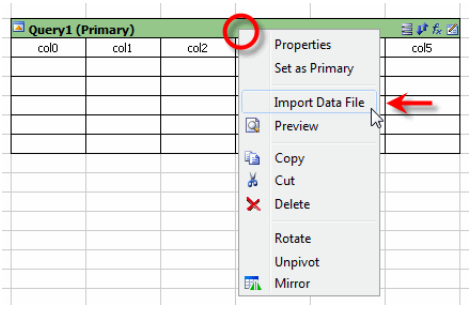
4. Press the 'New Embedded Table' button on the toolbar. This changes the standard pointer into a crosshairs pointer.



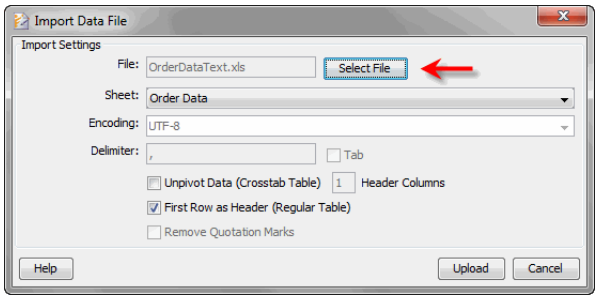
5. Drag the crosshairs pointer on the Worksheet grid to create a new table. (It does not matter how tall or wide you make the table.)



6. Right-click on the title bar of the new table, and select ‘Import Data File’. This opens the ‘Import Data File’ dialog box.



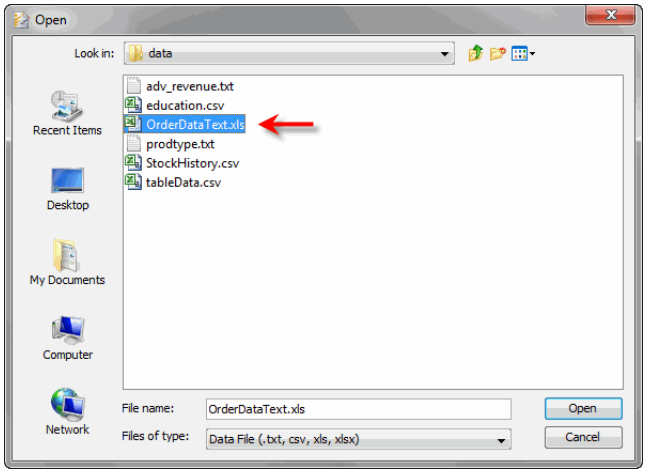
7. Press the ‘Select File’ button.



8. In the file browser, navigate to the following directory in the Style Scope installation folder:

   `{Style Scope}\examples\docExamples\datasource\data`

9. Select the *OrderDataText.xls* file.



10. Press ‘Upload’ to import the data. (Do not change the other settings.) This populates the table with data from the file.



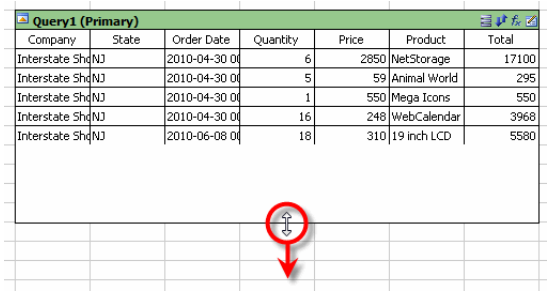
Query1 (Primary)

Company	State	Order Date	Quantity	Price	Product	Total
Interstate Sho	NJ	2010-04-30 0	6	2850	NetStorage	17100
Interstate Sho	NJ	2010-04-30 0	5	59	Animal World	295
Interstate Sho	NJ	2010-04-30 0	1	550	Mega Icons	550
Interstate Sho	NJ	2010-04-30 0	16	248	WebCalendar	3968
Interstate Sho	NJ	2010-06-08 0	18	310	19 inch LCD	5580

The number of columns in the table is automatically adjusted to display the number of columns in the Excel file.

Note: A table is also called a **data block**.

11. To display more rows in the table (also called a “data block”), drag down the bottom border of the table.

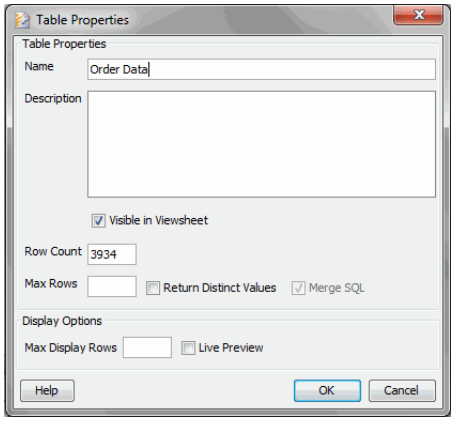


Query1 (Primary)

Company	State	Order Date	Quantity	Price	Product	Total
Interstate Sho	NJ	2010-04-30 0	6	2850	NetStorage	17100
Interstate Sho	NJ	2010-04-30 0	5	59	Animal World	295
Interstate Sho	NJ	2010-04-30 0	1	550	Mega Icons	550
Interstate Sho	NJ	2010-04-30 0	16	248	WebCalendar	3968
Interstate Sho	NJ	2010-06-08 0	18	310	19 inch LCD	5580

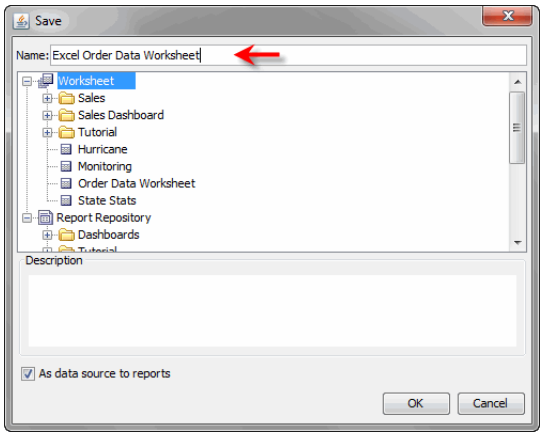
12. Right-click on the table, and select ‘Properties’ from the context menu. This opens the ‘Table Properties’ dialog box.

13. In the ‘Name’ field, enter the label “Order Data”, and press ‘OK’.



14. Press the 'Save' button in the toolbar. This opens the 'Save As' dialog box.

15. In the 'Name' field, enter "Excel Order Data Worksheet". Select the 'Worksheet' node on the tree, and press 'OK'.



This saves the Data Worksheet under the name "Excel Order Data Worksheet" at the root level.

You have now finished importing your data into the Data Worksheet, and you can proceed to create dashboards or reports based on this Data Worksheet. See *Creating a Dashboard* or *Creating a Report* for further instructions.

Because this data was imported directly into the Data Worksheet (rather than accessed through a *datasource*, as described in *Connecting to Text or Excel Data*), changes that you subsequently make to the Excel file will **not** be reflected in the Data Worksheet or the dashboards and reports based on the Data Worksheet. To update the Data Worksheet with new data, you will need to import the data again.

Creating a Query with the Wizard

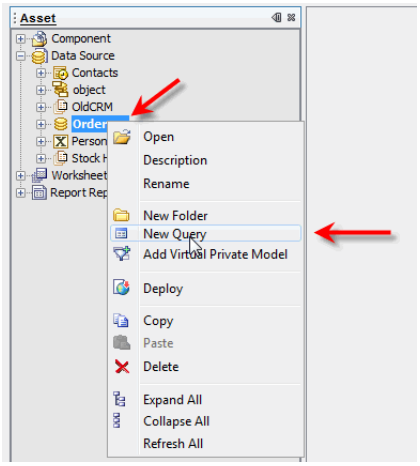
After you have connected to your data (see *Connecting to Text or Excel Data* and *Connecting to a Database*), you can create a query to retrieve your data.

When you create the first query for a data-source, Style Studio imports and caches all meta-data (table and column names and types), which may take a few minutes.

To create a new query using the Query Wizard, follow the steps below:

1. Open the Style Studio **Asset** panel, and expand the 'Data Source' node.

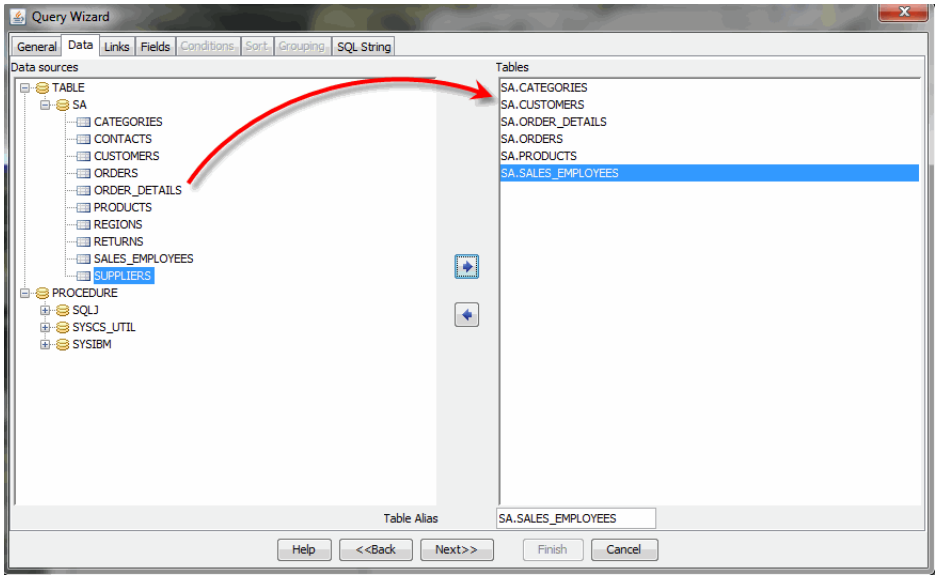
2. Right-click on the data source that you wish to query, and select 'New Query' from the context menu. This opens the Query Wizard.



3. Under the **General** tab of the Query Wizard, enter a name for the query and (optional) description. Press 'Next' to advance to the next tab.

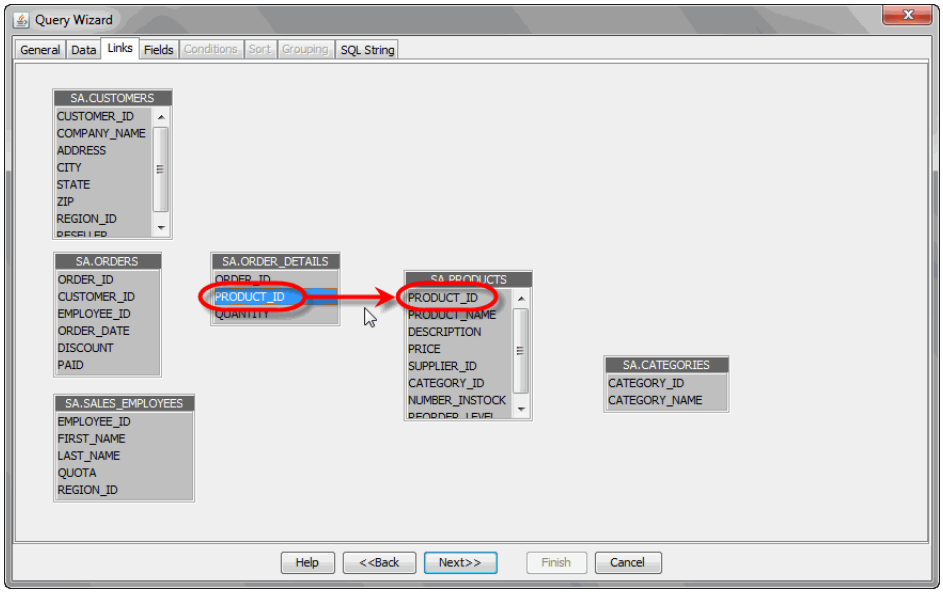
You can also double-click fields to add them, or use the arrow buttons.

4. Under the **Data** tab, drag the desired database tables from the 'Data Sources' panel to the 'Tables' panel. Press 'Next' to advance to the next tab.

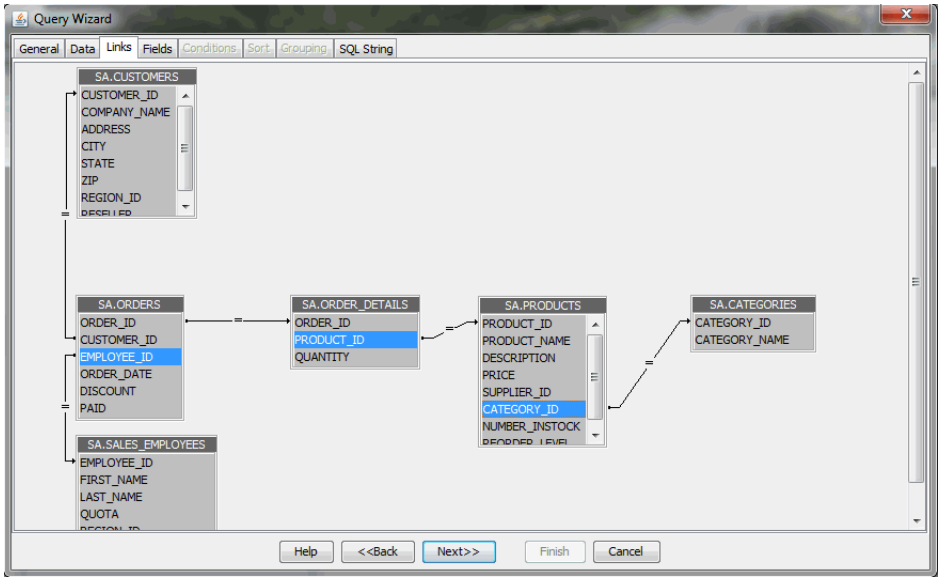


To manually enter SQL rather than using the Wizard, press the **SQL String** tab. See *Creating a Query from SQL*.

5. Under the **Links** tab, define a set of joins to relate the tables. (In some cases, a default set of joins may have been created automatically.) To create a join between different tables, simply drag a field from the first table onto the related field in the second table.

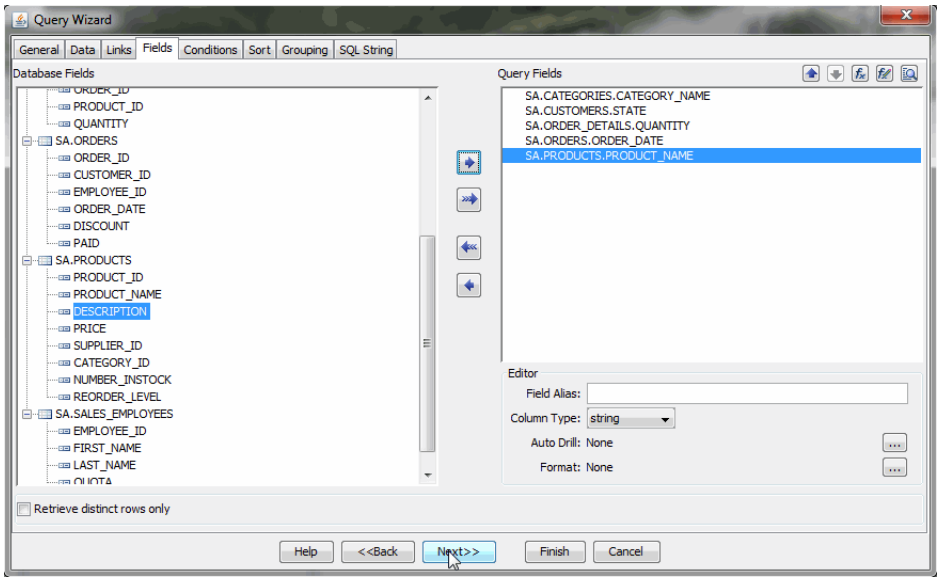


When you have joined all the tables, press ‘Next’ to advance to the next tab.

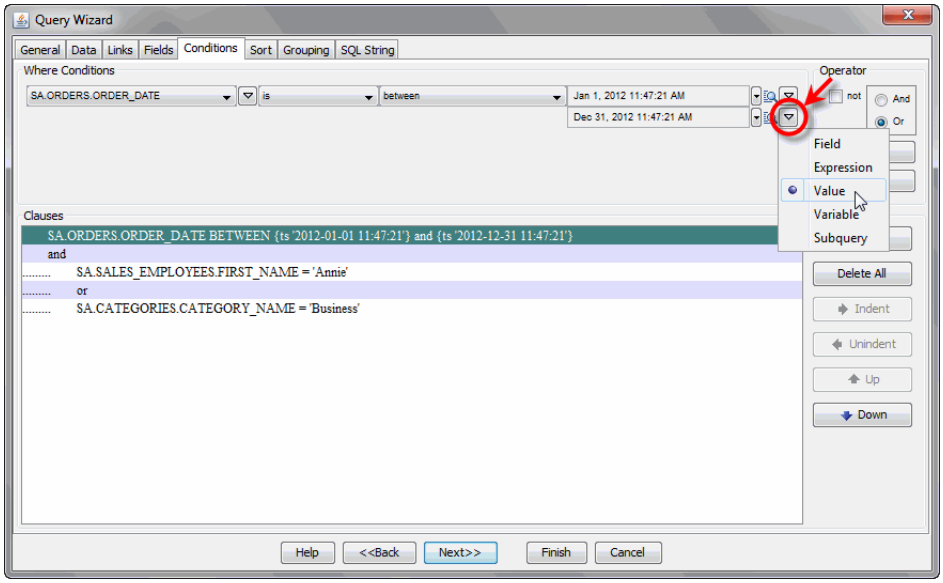


You can also double-click fields to add them, or use the arrow buttons.

6. On the **Fields** tab, select the fields that you want the query to return. To do this, drag the desired fields from the 'Database Fields' panel to the 'Query Fields' panel. Then press 'Next' to advance to the next tab.



7. On the **Conditions** tab, use the menus to specify a filtering condition. This condition will restrict the data returned by the query.

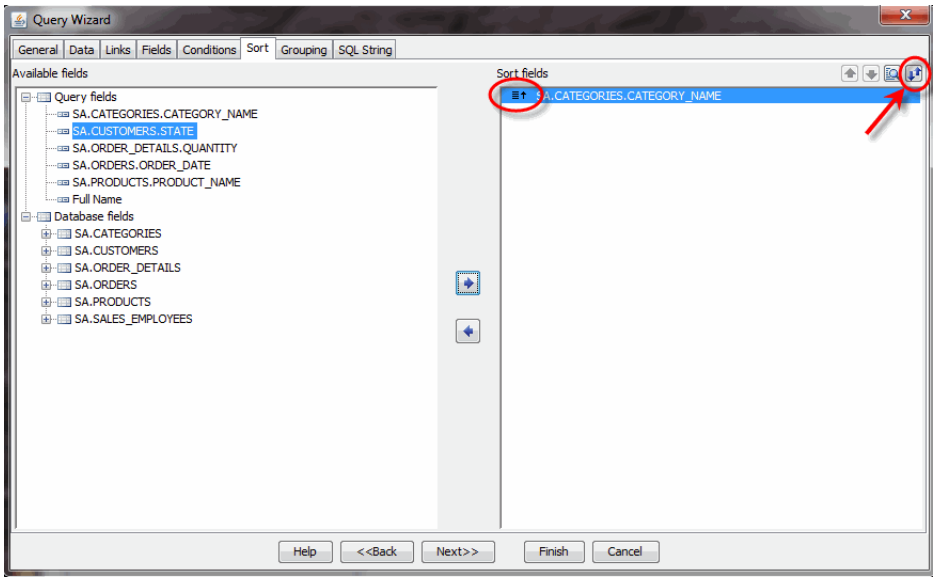


Press ‘Next’ to proceed to the **Sort** tab.

8. On the **Sort** tab, select the fields on which you want the query to be sorted. To do this, drag the desired fields from the ‘Available Fields’ panel to the ‘Sort Fields’ panel.

An arrow to the left of the field name indicates the current sort order.

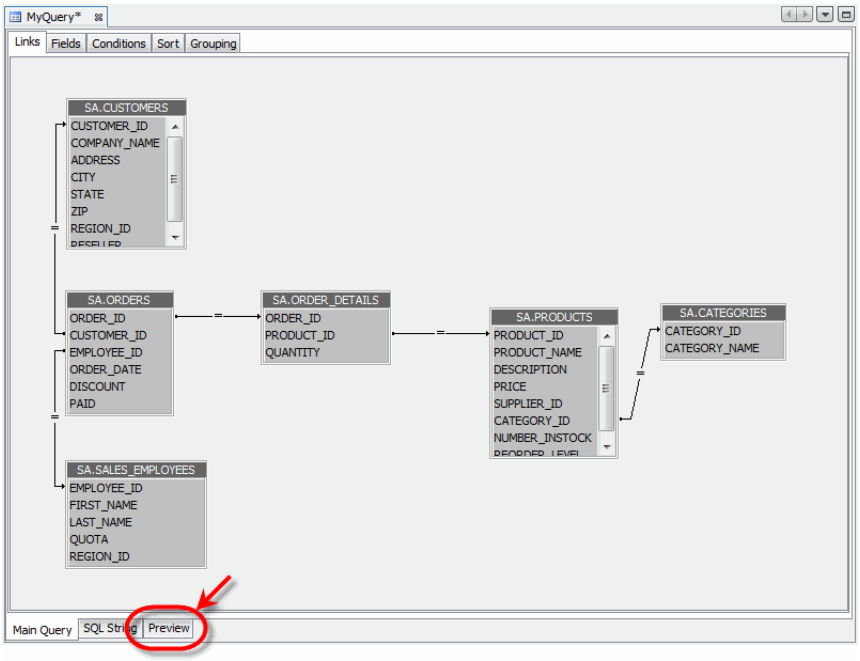
9. Select a field in the ‘Sort Fields’ panel, and press the ‘Order’ button above the panel. Press once to sort in descending order. Press again to sort in ascending order.



10. Press ‘Finish’ to complete the Query Wizard. This displays the query definition screen.

11. Press the ‘Save’ button in the toolbar to save the query. The query is now listed under the data source in the Style Studio **Asset** panel, and will be available for use in dashboards and reports.

12. To execute the query and preview the result set, select the **Preview** tab at the bottom of the query definition screen.



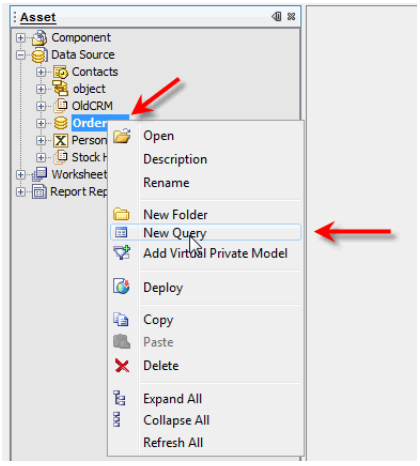
Creating a Query from SQL

After you have connected to your data (see *Connecting to Text or Excel Data* and *Connecting to a Database*), you can create a query to retrieve your data.

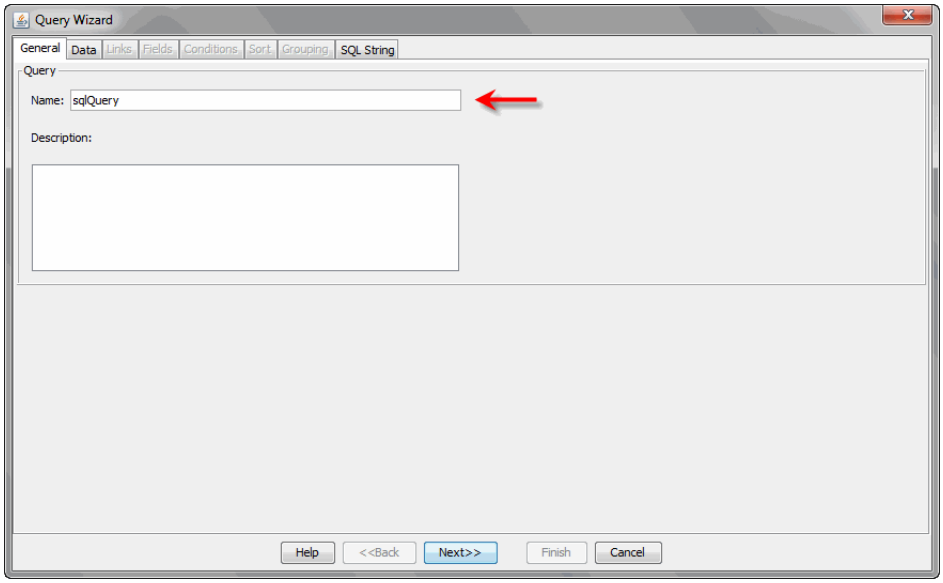
If you would like to use an existing SQL query string, follow the steps below:

1. Open the Style Studio **Asset** panel, and expand the 'Data Source' node.

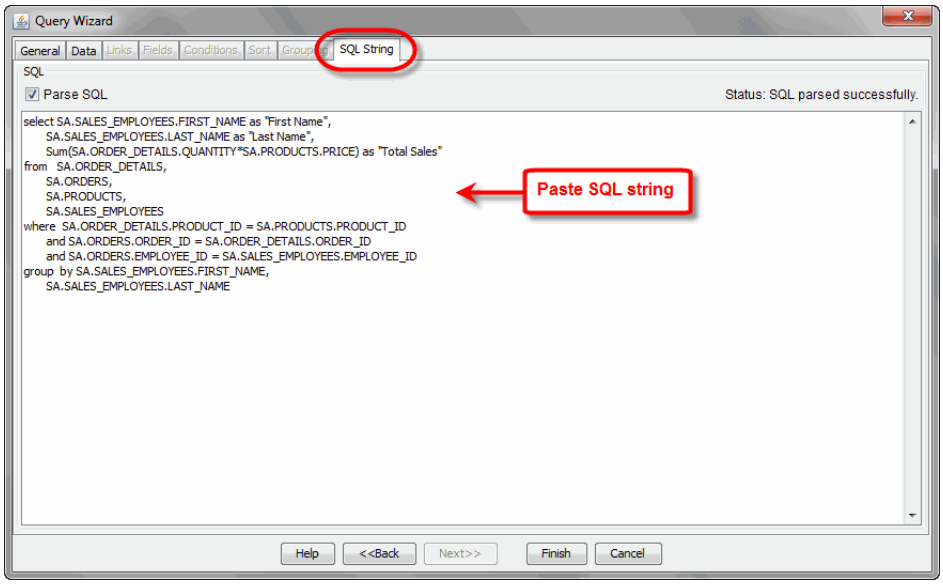
2. Right-click on the data source that you wish to query, and select 'New Query' from the context menu. This opens the Query Wizard.



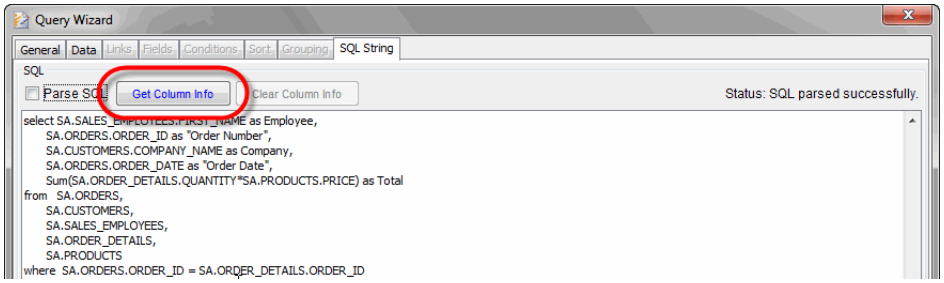
3. Under the **General** tab of the Query Wizard, enter a name for the query and (optional) description.



4. Select the **SQL String** tab. Paste the existing SQL string into the text field. If the SQL is very complex or uses database-specific syntax, uncheck the 'Parse SQL' option.



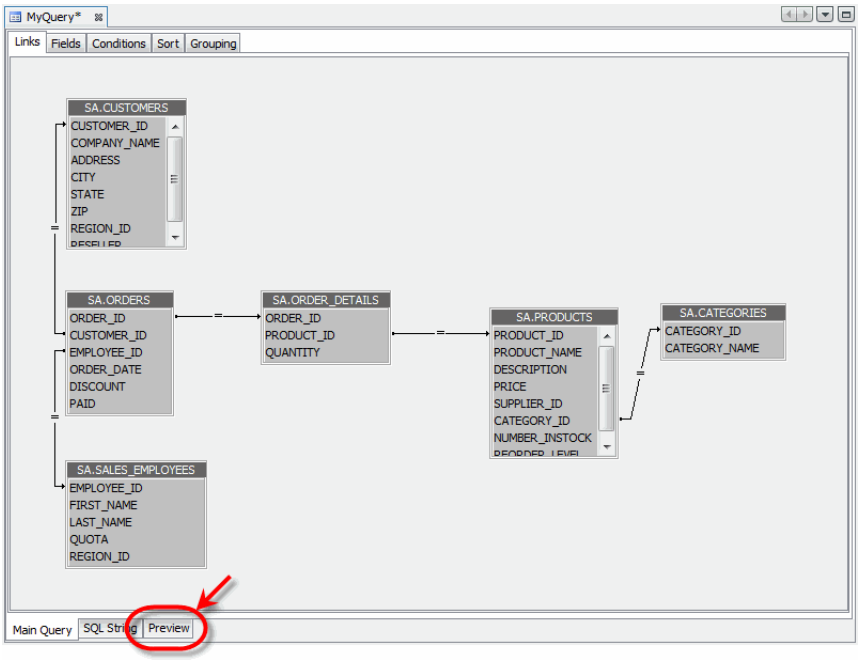
If you uncheck the 'Parse SQL' option, press the 'Get Column Info' button to acquire the necessary meta-data.



5. Press 'Finish' to complete the Query Wizard. This displays the query definition page, including a graphical representation of the query you entered (if the 'Parse SQL' option is enabled).

6. Press the 'Save' button in the toolbar to save the query. The query is now listed under the data source in the Style Studio **Asset** panel, and will be available for use in dashboards and reports.

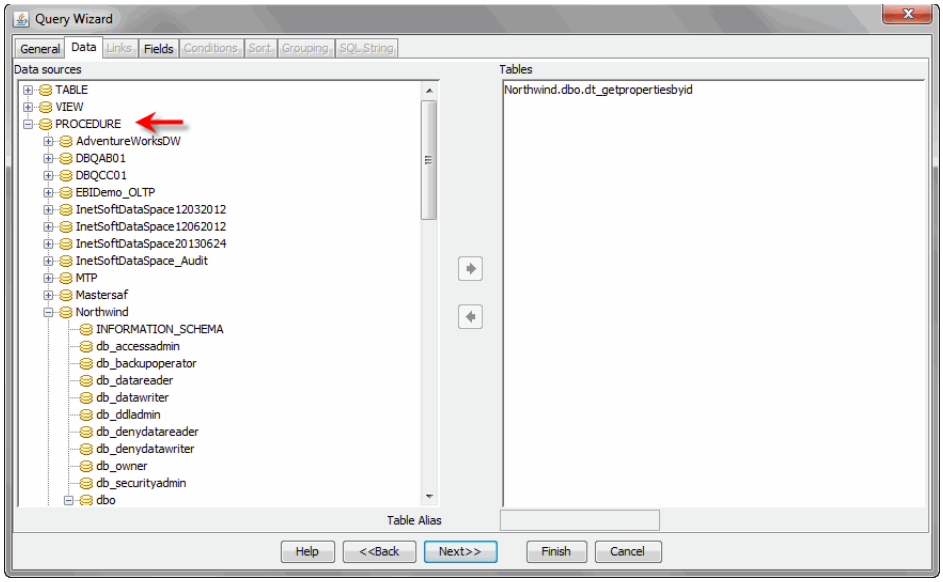
7. To execute the query and preview the result set, select the **Preview** tab at the bottom of the query definition screen.



Creating a Query from Stored Procedure

After you have connected to your data (see *Connecting to Text or Excel Data* and *Connecting to a Database*), you can create a query to retrieve your data.

To access a stored procedure, follow the steps in *Creating a Query with the Wizard*. Under the **Data** tab, select the desired stored procedure from the 'PROCEDURE' node on the tree, and proceed to select the desired fields on the **Fields** tab.



There are certain restrictions on the use of stored procedures:

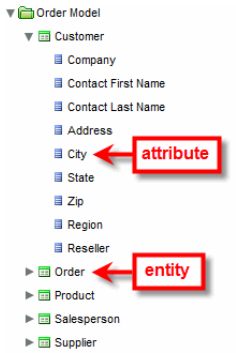
- Only one dataset is allowed in a stored procedure.
- In Oracle, only one cursor is allowed and it must be the last parameter in the stored procedure.
- In Oracle, only standard SQL types should be used in the parameters (user-defined types are not allowed).
- In PostgreSQL, create a free-form SQL query of the following form:

```
select * from procedureName(parameters)
```

- In MySQL, older drivers do not return procedure meta-data.

Creating a Data Model

A *data model* is a representation of the underlying database schema. The data model reorganizes the database tables and columns to provide a simplified entity-attribute interface for users who are designing reports and dashboards. For example, the sample data model called 'Order Model' organizes attributes into a set of five entities, 'Customer', 'Order', 'Product', 'Salesperson', and 'Supplier'.

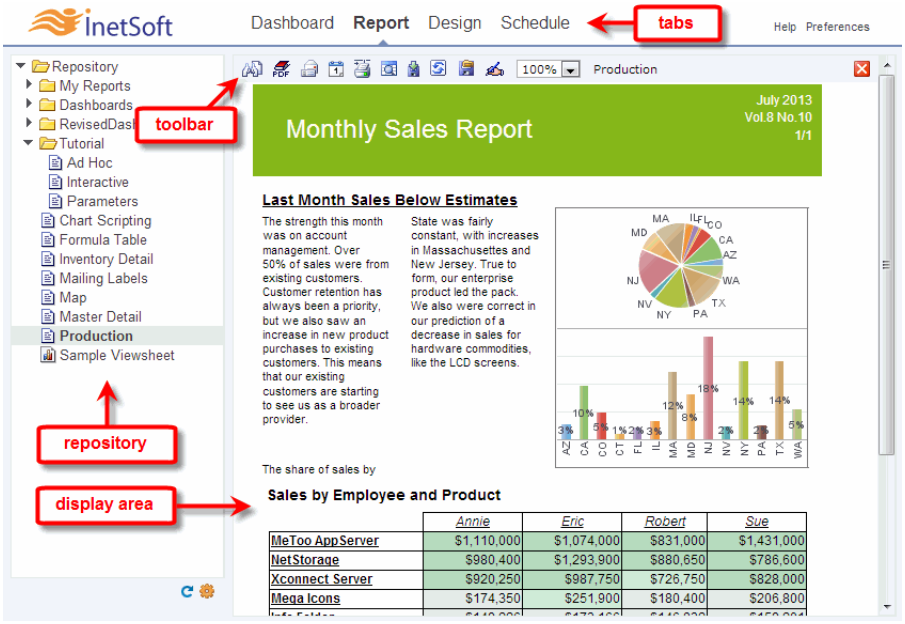


For information on how to create a data model, see *Relational Databases* in *Advanced Data Modeling*.

Using the Portal

The User Portal provides features for accessing and managing existing Style Intelligence assets, such as reports and dashboards, and allows you to create new assets as well.

The User Portal is a web-based interface, so you do not need to install any special software on your computer. Your system administrator will tell you the correct address to use in your web browser.



There are three main areas of the User Portal:

- **Portal Tabs**: Provide quick access to dashboards, scheduling features, and design features.
- **Repository Tree**: Provides access to existing reports and dashboards.
- **Display Area**: Displays the requested reports and dashboards.

Starting the Server

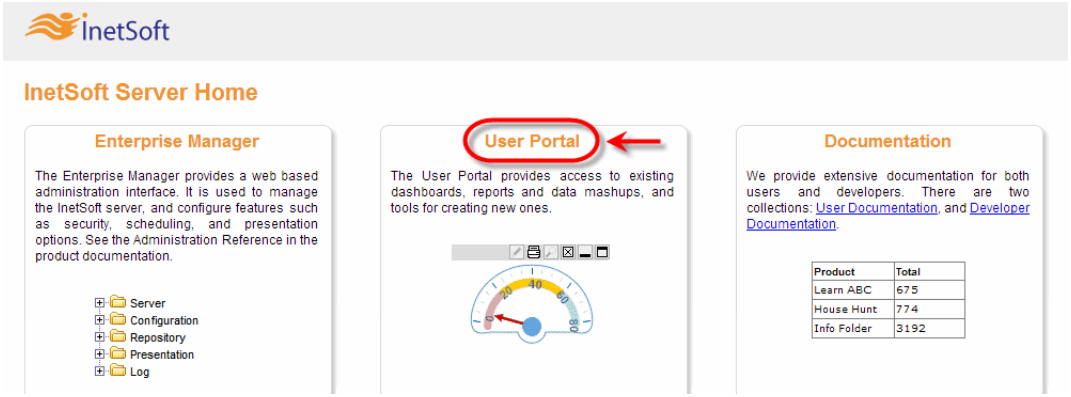
The Style Scope or Style Intelligence server must be running in order to access features of the User Portal and Enterprise Manager.

If the server is already running, simply enter the server URL in any browser. The default URL for a local server is `http://localhost:8080/sree/`.

If the server is not yet running, start the server as follows:

- On Windows, launch the server from the Windows 'Start' menu: *Style Scope* > *Style Scope Server* or *Style Intelligence* > *Style Intelligence Server*. This starts the server and opens the browser to the server home page.
- On Linux, run the *InetsoftServer.sh* file in the *bin* folder of the installation directory to start the server. In your browser, enter the URL `http://localhost:8080/sree/` to access the server home page.

Select the 'User Portal' link to view reports and dashboards. Select the 'Enterprise Manager' link to administer the server.

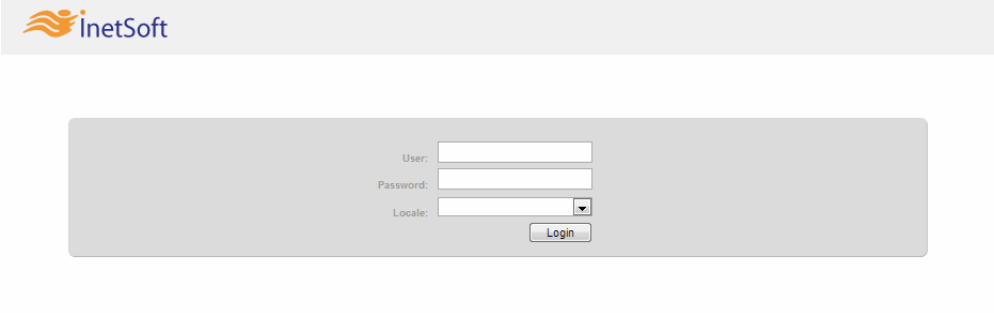


Logging into the Portal

If the Style Scope or Style Intelligence server is running, simply enter the server URL in any browser. Your administrator can provide this URL. (The default URL for User Portal on a local server is `http://localhost:8080/sree/Reports`.)

If the server is not running, see *Starting the Server*.

If security is enabled on the server, you must log in before you can use the Portal. Enter your username and password in the provided 'User' and 'Password' fields. (Ask your system administrator if you don't know your username and password.) If you are prompted to enter a 'Locale', select the correct option for your location.

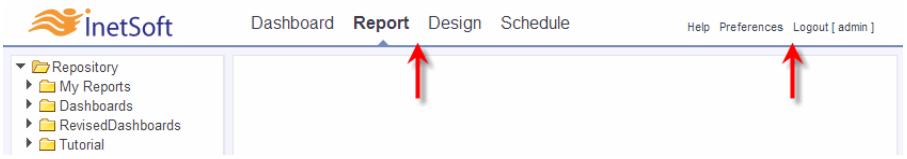


After you log in, select the desired Portal tab for the features you want to access or press the 'Help' link to get information about using the Portal. See *Accessing Portal Functions* for more information about the available features.

Note: For security reasons, login information expires after five minutes. If you receive a 'Login Expired' warning, simply re-enter your credentials.

Accessing Portal Functions

The User Portal contains tabs for viewing and creating Style Intelligence assets. The options available will depend on what permissions you have been granted by the administrator.

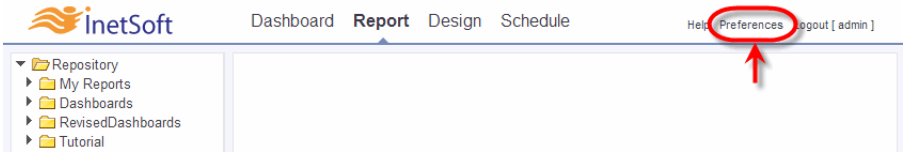


The table below explains the tabs and links that appear at the top of the Portal.

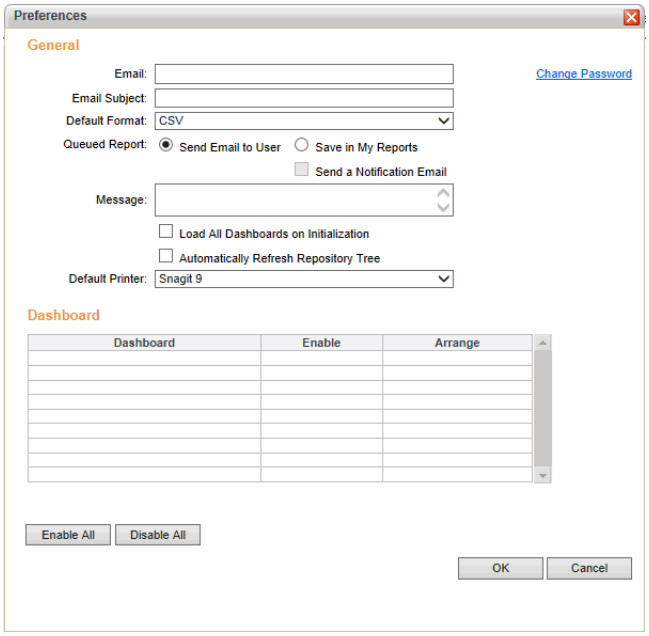
TAB	**FUNCTION**
Dashboard	Access your important dashboards. See *Deploying a Dashboard* for information on configuring dashboards.
Report	View all available dashboards and reports in the repository. See *Interacting and Collaborating* and *Using a Report* for more details on using dashboards and reports.
Viewer	View all available dashboards in the repository (Style Scope only). See *Interacting and Collaborating* for more details on using dashboards.
Design	Create and deploy dashboards. See *Creating a Dashboard* and *Deploying a Dashboard* for more information.
Schedule	Schedule dashboards and reports to generate automatically, and monitor queued and completed scheduled tasks. See *Scheduling Dashboards and Reports* for more details.
LINK	**FUNCTION**
Help	Open the user documentation.
Preferences	Change personal settings and configure user dashboards. See *Configuring Portal Preferences* for more information.
Log Out	Log out of the Portal (when security is enabled).
Search	Search within reports. Logic (AND, OR) and wildcards (*, ?) are allowed. For example, “wee*” will find “week” and “weekly.” See *Searching Archived Reports* for more information about archiving. To search an open report, see *Searching a Report*.

Configuring Portal Preferences

To manage user preferences in the Portal, press the 'Preferences' link at the top of the Portal.

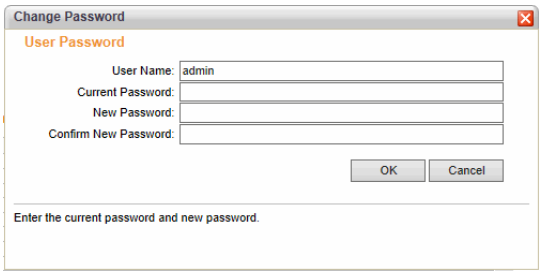


This opens the 'Preferences' dialog box.



The following preferences are available:

- **Change Password**: Press the 'Change Password' link to open the 'Change Password' dialog box. Enter your current password and a new password and press 'OK'.



- **Email**: Enter the email address to which queued reports and notifications should be sent.
- **Email Subject**: Enter the subject line to be used for queued report emails sent to the address specified in the ‘Email’ field. If nothing is specified for the ‘Email Subject’ field, the report name will be used as the subject text.
- **Default Format**: Select the default format in which queued reports should be emailed or saved.
- **Queued Report**: Select ‘Send Email to User’ to have queued reports delivered by email to the address specified in the ‘Email’ field. Select ‘Save in My Reports’ to save an archived version of the report in the ‘My Reports’ repository folder. (This requires archiving to be enabled by an administrator.) Enable ‘Send a Notification Email’ to receive a notification email when the report has finished generating. See *Running a Report* for information about queueing a report.
- **Message**: Enter a message to be displayed in the body of the email delivered to the address specified in the ‘Email’ field.
- **Load all dashboards on initialization**: Forces all enabled dashboards to reload when any dashboard is accessed. This is only needed when you have multiple dashboards that share a filter. (See *Sharing Filters* in the *Advanced Dashboard Design* for more information.) For all other cases, this option should remain disabled.
- **Automatically Refresh Repository Tree**: Allows the server to automatically update the listing of available reports and dashboards. You can also press the ‘Refresh’ button below the Repository panel to update the listing on-demand.
- **Default Printer**: Select the printer that you wish to use as your default Portal printer.

To enable and disable dashboards or to change the order of the dashboard tabs, follow these steps:

- In the ‘Dashboard’ table, select a checkbox in the ‘Enable’ column to make the corresponding dashboard visible in the **Dashboard** tab.
- Press the up-arrow and down-arrow buttons in the ‘Arrange’ column to change the order of the various dashboard tabs in the Portal.

See *Deploying a Dashboard* for information about adding a Viewsheet to the **Dashboard** tab.

See Also

Scheduling a Dashboard, for information about emailing or saving a dashboard automatically.

Deploying a Dashboard, to create a new dashboard under the ‘Dashboard’ tab.

Organizing the Repository

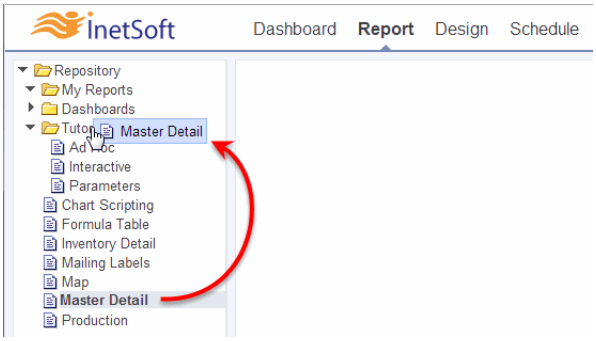
To create a new folder in the Repository, right-click on the folder that you want to contain the new folder, and select ‘New Folder’. This opens the ‘Create Folder’ page.



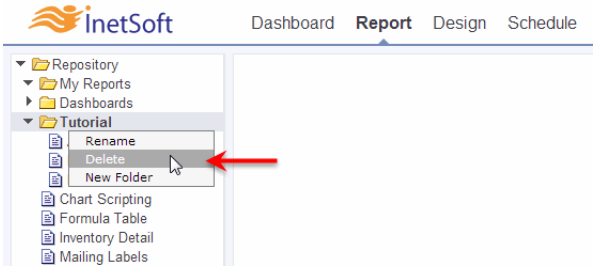
The ‘Alias’ is the name displayed in the Repository.

Enter the desired ‘Folder Name’, and (optionally) supply an ‘Alias’ and ‘Description’. Then press ‘Apply’. This creates a new folder in the specified location.

To move reports, dashboards, or folders from one Repository folder to another, simply drag the asset to the desired location.



To delete a report, dashboard, or folder from the Repository, right-click on the asset and select ‘Delete’.



To rename a report, dashboard, or folder, right-click on the asset and select ‘Rename’. Type the new name, and press ‘Enter’ on the keyboard or simply click somewhere else.

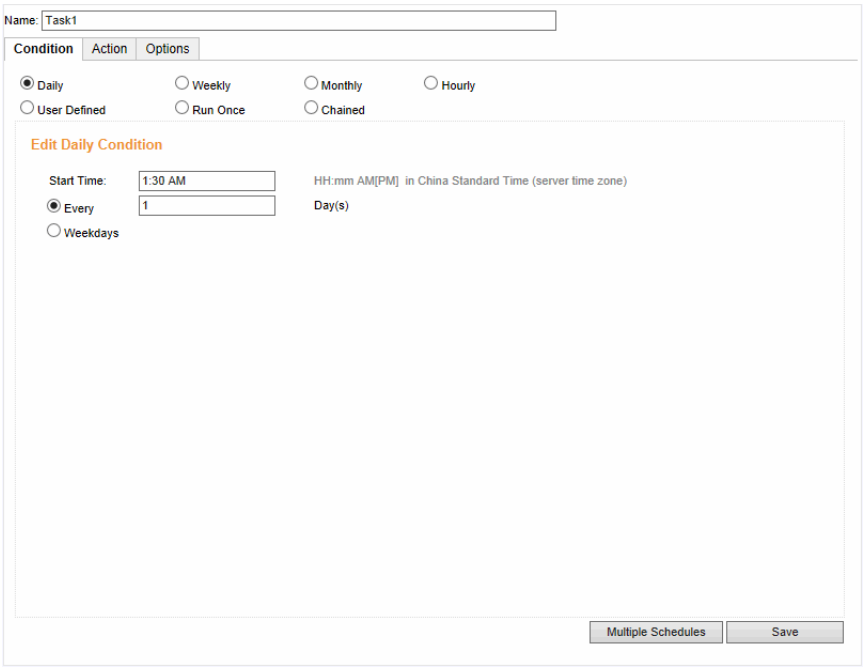
Scheduling Dashboards and Reports

To create a basic schedule, see *Scheduling a Dashboard* or *Scheduling a Report*.

To scheduled a report or a dashboard, follow the steps below.

1. Select the **Schedule** tab in the Portal.

2. Press the ‘New Task’ button below the Schedule table. This creates a new task with the name ‘Task1’, and opens the Editor.

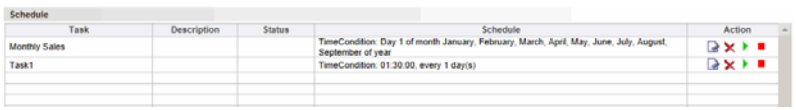


   Using the tabs at the bottom of the Editor, specify when the task executes, what actions it performs, and additional options. See *Setting Schedule Conditions*, *Setting Schedule Actions*, and *Setting Schedule Options* for more information.

3. Press ‘Close’ to exit the Editor.

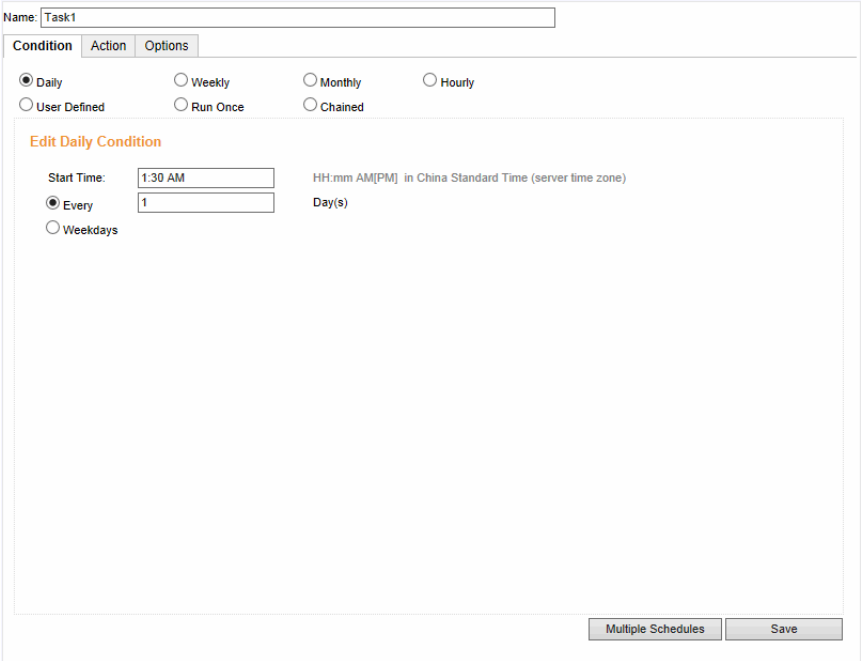


To modify an existing task, press the task’s ‘Edit’ button in the ‘Schedule’ table. This opens the Editor as described above.



Setting Schedule Conditions

A schedule condition is used to determine when the schedule task executes.



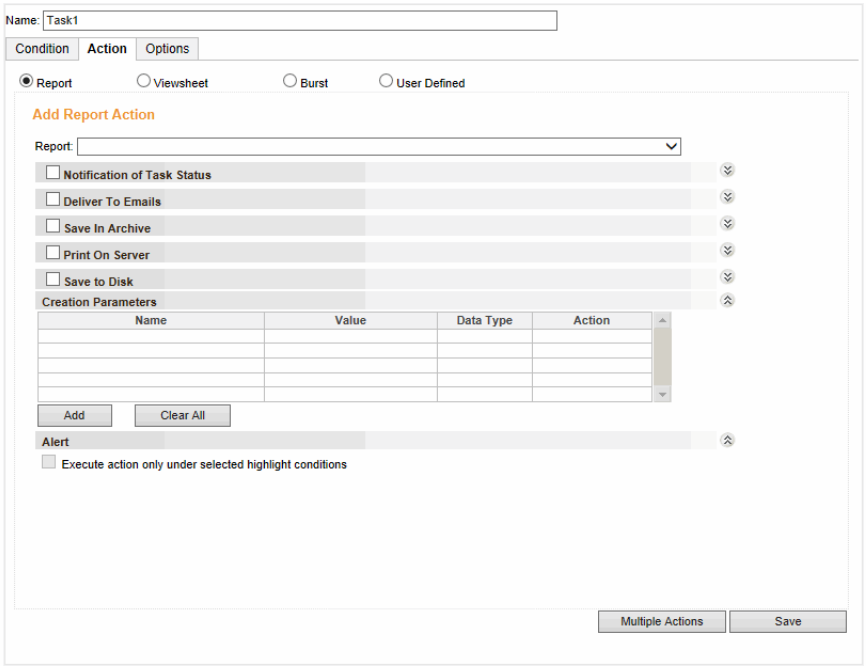
There are six types of conditions: Daily, Weekly, Monthly, Run Once, Chained, and User Defined.

- **Daily**: This task will execute every N days (every 1 day, every 5 days, etc.) at the specified time. The time is specified using the format HH:mm [am|pm], for example, “11:39 pm”. Select the ‘Weekdays’ option to execute the task at the specified time on weekdays only.
- **Weekly**: This task will execute every N weeks (every 1 week, 4 weeks, etc.) at the specified time and days of the week.
- **Monthly**: This task will execute every Nth day of the specified months (e.g., April 15) or every Nth day of the week of the specified month (e.g., 1st Sunday of May).
- **Run Once**: This task will execute once on a certain day at a specified time (e.g, Nov. 25, 2006 at 11:30 am).
- **Chained Condition**: This task will execute only if another specified task or cycle has successfully completed first.
- **User Defined Condition**: ‘User Defined’ conditions (if any) that have been created by a developer within your organization can be selected from the menu.

To specify multiple conditions, press the ‘Multiple Schedules’ button at the bottom right of the editing panel. This will display the schedule condition list to which you can add, delete or edit conditions.

Setting Schedule Actions

An “action” specifies the set of operations to be performed when the scheduled task runs. There are three types of actions: ‘Report’, ‘Dashboard’, ‘Burst’, and ‘User Defined’.



The **Report Action** executes a report and delivers or saves it. The options for report action are shown below.

OPTION	DESCRIPTION
Save in Archive	Save the generated report in the report archive. A report archive must be properly configured, and you must have proper permissions to save the report in the selected folder.
Print on Server	Print to a server printer.
Deliver to Emails	Send a report to other users via email. To embed the report within the email as HTML, select the ‘HTML Email’ option. To send the report as an attachment, select one of the other formats (PDF, Excel, etc.). If the attached file is too large, it will be split into multiple emails. The email subject line can include parameters for automatic insertion of report alias and time/date of report generation. See below for examples. The ‘Bundled as zip’ option allows you to zip the attachment and optionally specify a password to encode the archive with WinZip 256-bit AES encryption.

OPTION	DESCRIPTION
Save to Disk	Save the report on the local file system in any of the following formats: PDF, HTML, Excel, RTF, SVG, CSV, Text, XML. The 'Path' field should specify a valid absolute path, including filename. (The filename extension is added automatically.) The filename can include parameters for automatic insertion of report name and time/date of report generation. See below for examples.
Notify when Completed	Send users an email notification about the task completion status. Optionally you can check 'Notify only if failed'.

The 'Deliver to Emails' subject line uses the java.text.MessageFormat syntax with two properties. The first (index 0) is the report alias and the second (index 1) is the date and time of report generation. For example, the 'Subject' specification

```
Report named {0}, generated at {1,time} on {1,date}
```

would produce (for a report with alias 'testReport') an email subject line such as the following:

```
Report named testReport, generated at 10:34:54 AM on Nov 5,
2007
```

See below for some examples of date formatting using the java.text.SimpleDateFormat. The report alias is set by an administrator, and may often be the same as the report name.

The filename for the 'Save to Disk' action and the attachment name for the 'Deliver to Emails' action can include the same parameters as the email subject line. However, because of filename restrictions, you cannot use the "`{1,time}`" syntax. Some examples of automatically generated filenames are shown below.

SPECIFIED FILENAME	GENERATED FILENAME
myRep_{1,date}.pdf	myRep_Nov 14, 2007.pdf
{0}_{1,date}.pdf	"report alias"_Nov 14, 2007.pdf
myRep_{1,date,MMM-dd}.pdf	myRep_Nov-14.pdf
myRep_{1,date,EEE-h-mm a}.pdf	myRep_Wed-11-49 AM.pdf
myRep_{1,date,yy-MM-dd-HH-z}.pdf	myRep_07-11-14-11-EST.pdf

The **Dashboard Action** allows you to schedule the execution of a Viewsheet (dashboard), and email a notification or an exported copy of the dashboard to specified users. See 'Report Action' above for more details about these actions.

The **Burst Action** delivers multiple versions of a specified report, tailoring each version to the particular recipient. To do this, the report must first be

configured for bursting. See *Report Bursting* in *Advanced Report Design* for more details.

The **User Defined Action** allows a you to use a custom action class, if one exists.

To specify multiple actions, press the 'Multiple Actions' button at the bottom right of the editing panel. This will display the schedule action list to which you can add, delete or edit actions.

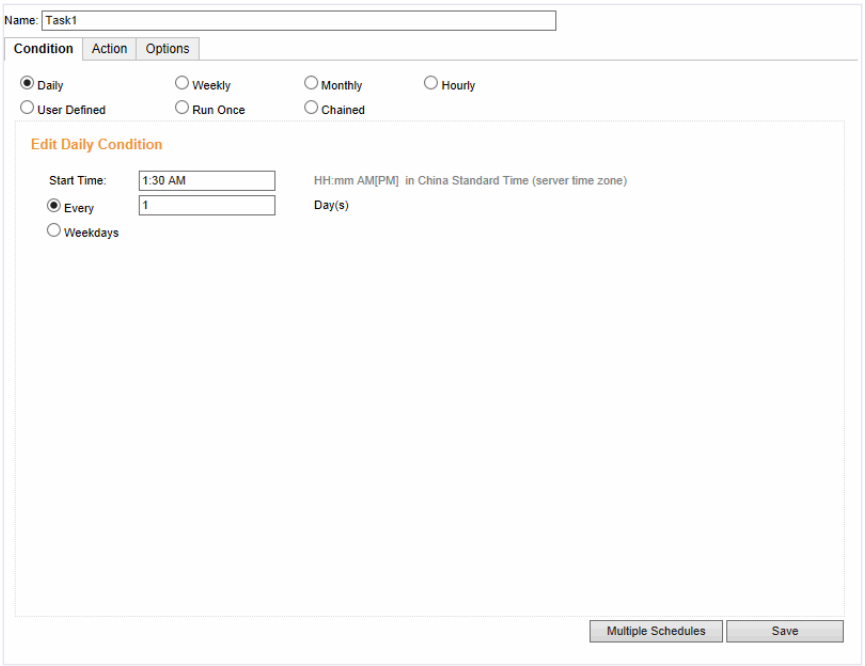
Setting an Alert

An alert is a notification that a specified event has occurred, for example, that total sales have fallen below some critical level. You can set an alert based on highlight conditions in a report or Viewsheet, or based on Viewsheet Output component ranges.

To configure an alert for a report or Viewsheet, follow the steps below:

1. Select the **Schedule** tab in the Portal.

2. Press the 'New Task' button below the Schedule table. This creates a new task with the name 'Task1', and opens the Editor.

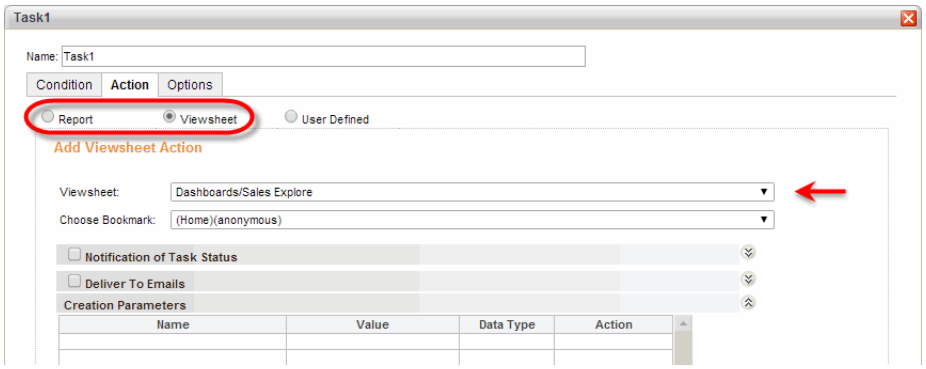


3. Change the task name as desired in the 'Name' field.

4. On the **Condition** tab, set the frequency with which the alert conditions should be tested. (See *Setting Schedule Conditions* for more details on the available day and time conditions.) For

example, if you want to test a report highlight condition every day, specify a ‘Daily’ condition.

5. On the **Action** tab, specify the report or Viewsheet that contains the alert condition you want to test.

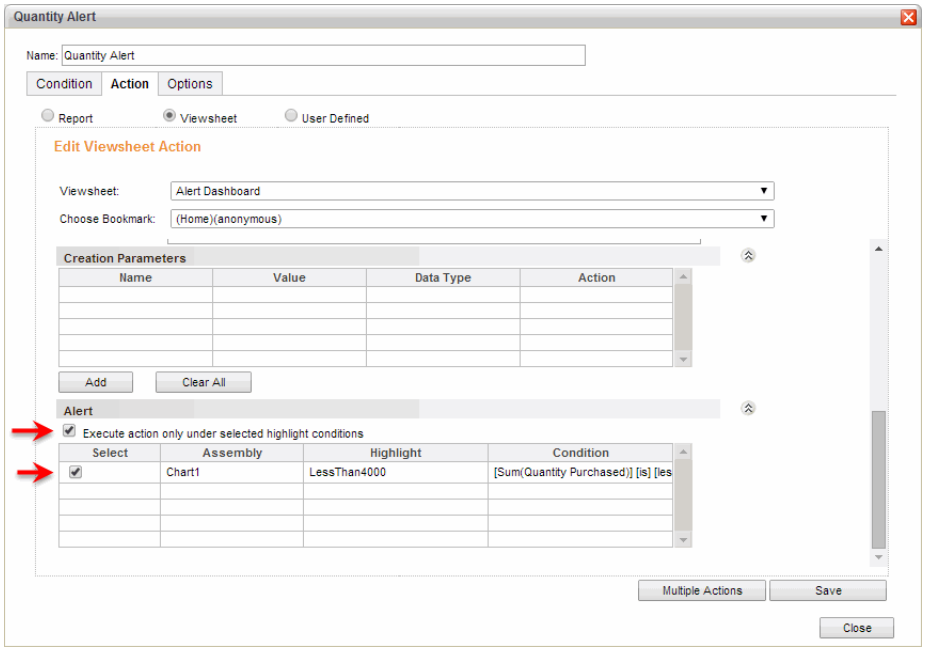


6. Select the actions that you want to perform if the alert is triggered. Generally, you will at least select the ‘Deliver To Emails’ option. (See *Setting Schedule Actions* for more about the available actions.) Enter a ‘Subject’ and ‘Message’ appropriate for the nature of the alert.

The table lists highlight conditions for all chart, table, image, and text components, as well as range conditions for Viewsheet Output components.

7. In the ‘Alert’ panel at the bottom of the **Action** tab, check the ‘Execute action only under selected highlight conditions’ option. This displays a list of the available highlight and range conditions for the selected Viewsheet or report.

8. In the table, select the highlight or range conditions that you want to trigger the alert. When *any* of the selected conditions test true, the actions you specified above (e.g., emailing) will be performed.



9. Press ‘Save’ to save the scheduled task. Then press ‘Close’ to close the dialog box.

See Also

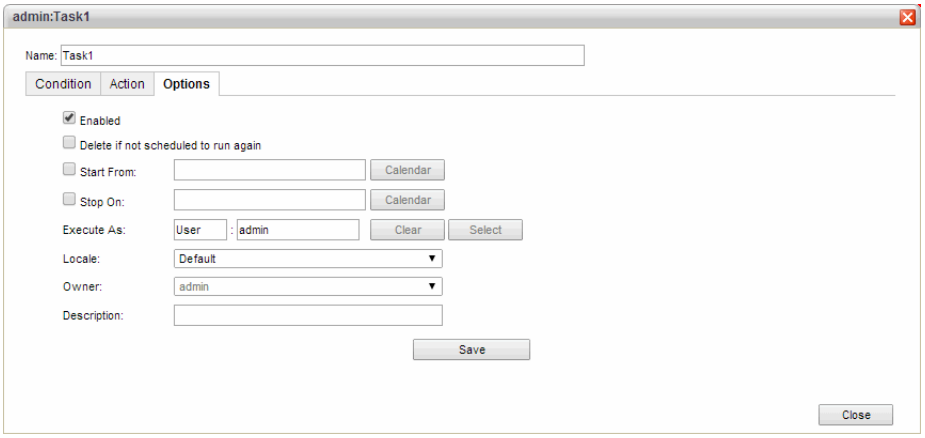
Highlights, in *Advanced Dashboard Design*, for information on creating Viewsheet highlights.

Highlighting, in *Advanced Report Design*, for information on creating report highlights.

Output Component Properties: Advanced Tab, in *Advanced Dashboard Design*, for information on adding ranges to Output components.

Setting Schedule Options

There are several options available when modifying a schedule task.

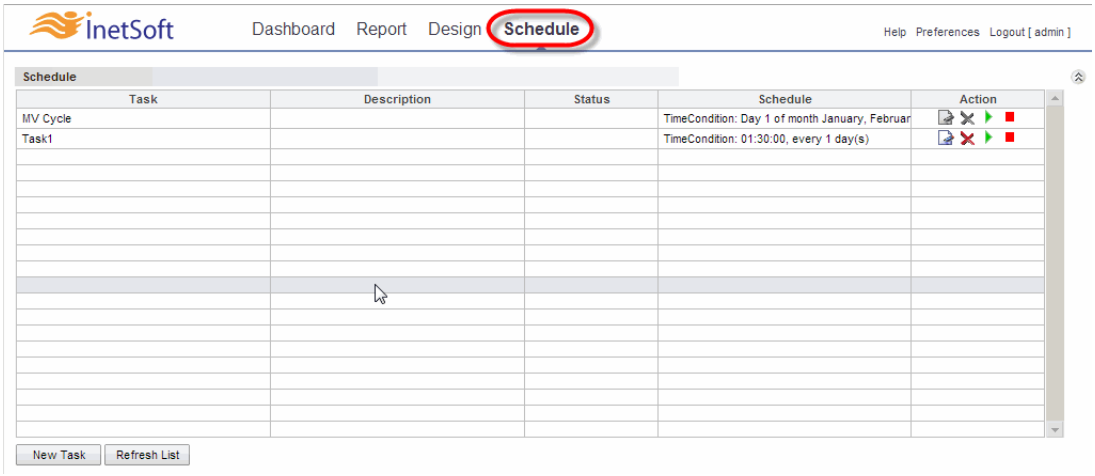


- **Enabled**: Temporarily enables or disables a task.

- **Delete if not scheduled to run again**: For a task that is scheduled to run only once, deletes the task from the system upon completion.
- **Start From and Stop On**: Execute the task only within the specified date range.
- **Execute As**: Execute the task as a particular user so that user permissions (e.g., VPM filtering) can be applied during task execution. Any user who has permission to use the Scheduler can be selected from the menu.
- **Locale**: Sets the locale for the scheduled task, which configures language and formatting.

Managing Scheduled Tasks

You can view and monitor all of your scheduled tasks from the **Schedule** tab.



This tab displays a list of the schedule tasks, showing start time, end time, status of the last run, and start time for the next run.



- Press the 'Edit' button to modify the task specifications. See *Setting Schedule Conditions*, *Setting Schedule Actions*, and *Setting Schedule Options* for more information.



- Press the 'Run Now' button to execute the task immediately.



- Press the 'Stop Now' button to stop the task immediately.



- Press the 'Delete' button to delete the task.

To temporarily disable a task, deselect the 'Enabled' checkbox on the **Options** tab. See *Setting Schedule Options* for more information.

Transforming Your Data

If you need to process your data, for example by doing filtering, grouping, or data mashup, read the following sections.

Creating a Data Worksheet

A Data Worksheet allows you to do advanced data transformation operations such as grouping, post-aggregate filtering, and data mashup. You can create and edit a Data Worksheet either in Style Studio or in the User Portal. The following sections explain how to do this.

Creating a Worksheet in Style Studio

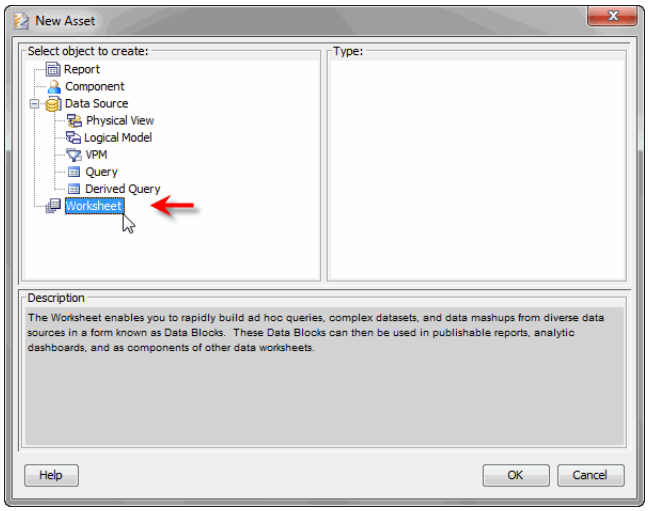
To create a new Data Worksheet in Style Studio, follow the steps below:

1. From the Windows *Start* Menu, select the 'Style Studio' shortcut in the 'Style Scope' program group. This starts Style Studio, the desktop development tool.

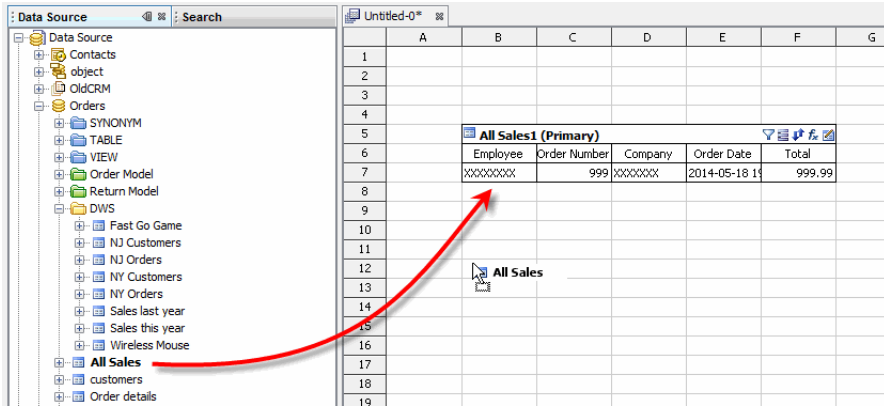


2. Press the 'New' button in the Style Studio toolbar. This opens the 'New Asset' dialog box.

3. In the left panel, select 'Worksheet', and press 'OK'. This opens a new Data Worksheet.



4. To add data blocks (tables) to the Data Worksheet, drag the desired data (queries, fields, or tables) from the **Asset** panel.



Proceed to the following sections to learn how to filter, group, and mash-up these data blocks.

See Also

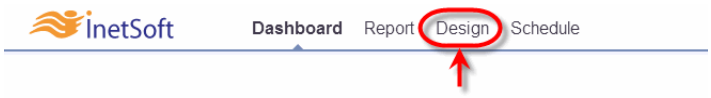
Creating a Data Table, in *Advanced Data Mashup*, for full information on hot to create data bocks.

Creating a Worksheet in the User Portal

To create a new Data Worksheet in the User Portal, follow the steps below:

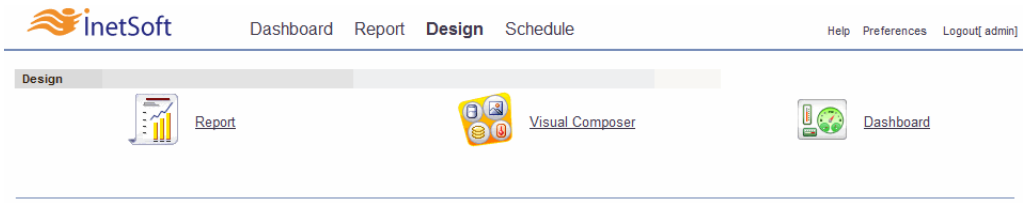
See *Using the Portal* for more information about the User Portal.

1. Log into the User Portal as described in *Logging into the Portal*.
2. Select the **Design** tab at the top of the User Portal.

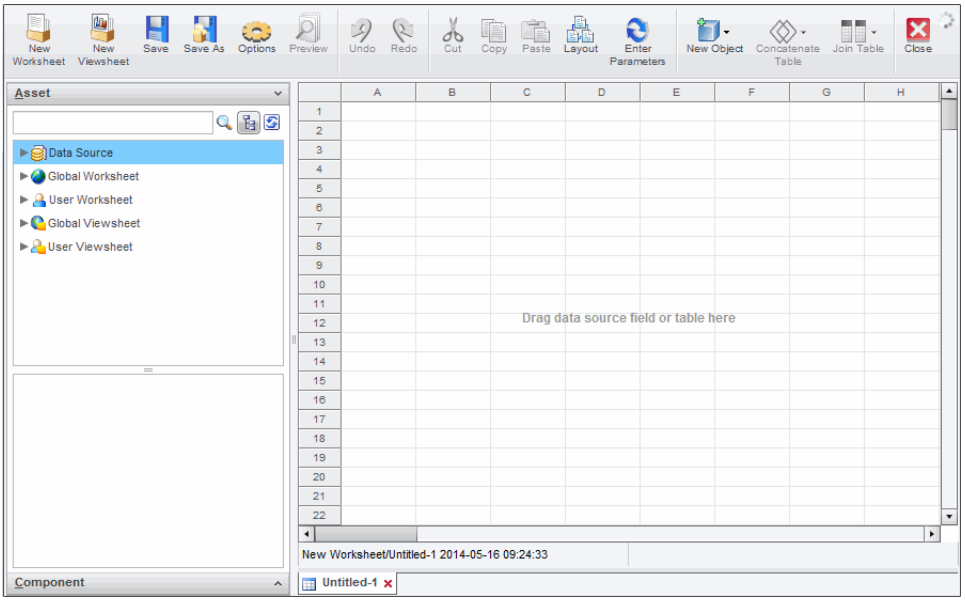


Note: Style Report EE does not provide access to Visual Composer. See *Creating a Worksheet in Style Studio*.

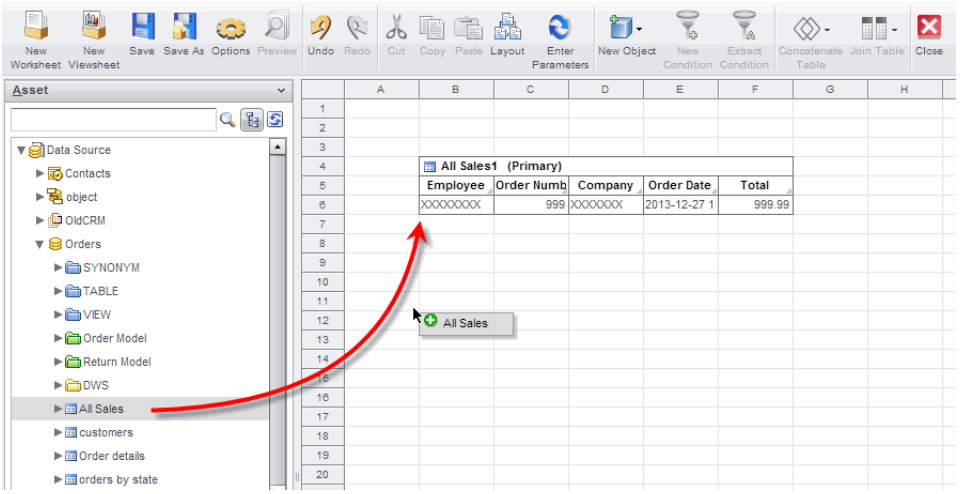
3. Press the 'Visual Composer' link under the **Design** tab. This launches the Visual Composer in a separate browser window.



4. Press the 'New Worksheet' button in the toolbar. This opens a blank Worksheet named 'Untitled'.



5. To add data blocks (tables) to the Data Worksheet, drag the desired data (queries, fields, or tables) from the **Asset** panel.



Proceed to the following sections to learn how to filter, group, and mash-up these data blocks.

See Also

Creating a Data Table, in *Advanced Data Mashup*, for full information on hot to create data bocks.

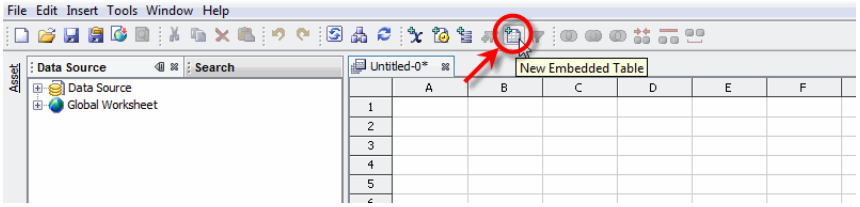
Importing a Data File

In most cases, data sources will already be configured for you by an administrator. However, you can also import a plain text file, CSV file, or Excel file into a Data Worksheet. In this example, you will import a sample Excel file.

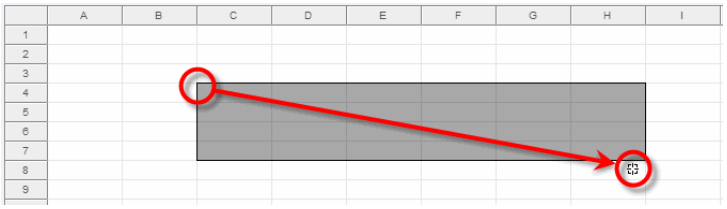
Follow the steps below to import the sample Excel file into a Data Worksheet.

1. Create a new Data Worksheet. See *Creating a Data Worksheet*.

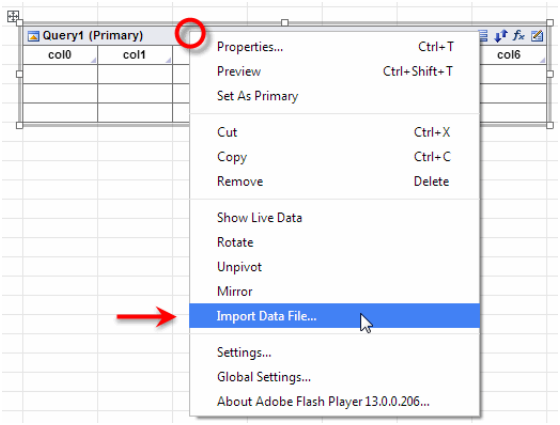
2. Press the 'New Object' button on the toolbar, and select 'Embedded Table'. This changes the standard pointer into a crosshairs pointer.



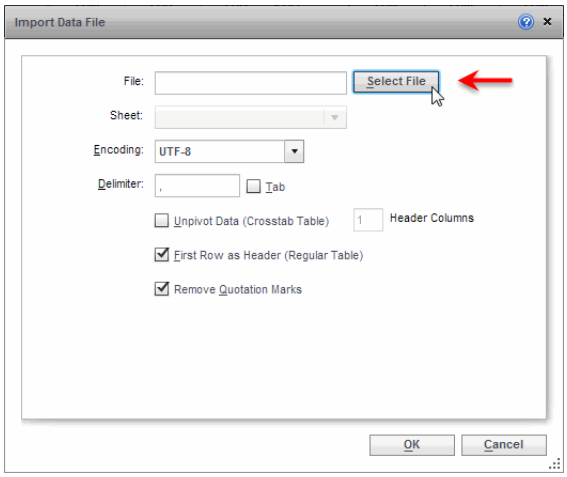
3. Drag the crosshairs pointer on the Worksheet grid to create a new table. (It does not matter how tall or wide you make the table.)



4. Right-click on the title bar of the new table, and select 'Import Data File'. This opens the 'Import Data File' dialog box.



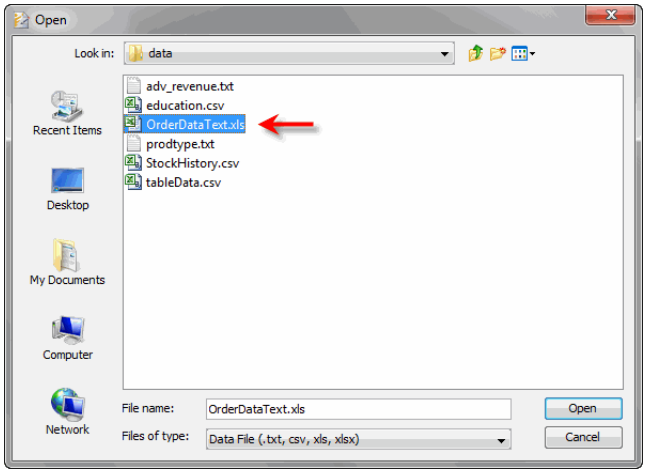
5. Press the 'Select File' button.



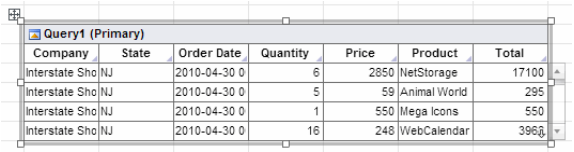
6. Select the Excel file you want to import.

   If you have Style Scope or Style Intelligence installed on your own machine, you can find the sample *OrderDataText.xls* Excel file in the following location:

   `{Installation}\examples\docExamples\datasource\data`



7. Press ‘Upload’ to import the data. (For the sample *OrderDataText.xls* Excel file, you do not need to change the other settings.) This populates the table with data from the file.



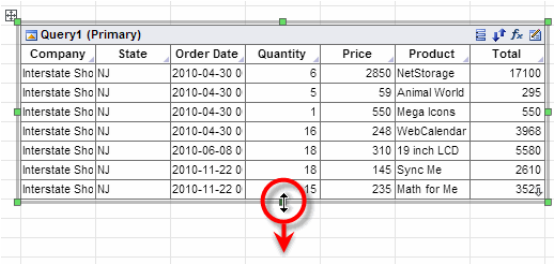
Query1 (Primary)

Company	State	Order Date	Quantity	Price	Product	Total
Interstate Sho	NJ	2010-04-30 0	6	2850	NetStorage	17100
Interstate Sho	NJ	2010-04-30 0	5	59	Animal World	295
Interstate Sho	NJ	2010-04-30 0	1	550	Mega Icons	550
Interstate Sho	NJ	2010-04-30 0	16	248	WebCalendar	3968

The number of columns in the table is automatically adjusted to display the number of columns in the Excel file.

Note: A table is also called a **data block**.

8. To display more rows in the table (also called a “data block”), drag down the bottom border of the table.



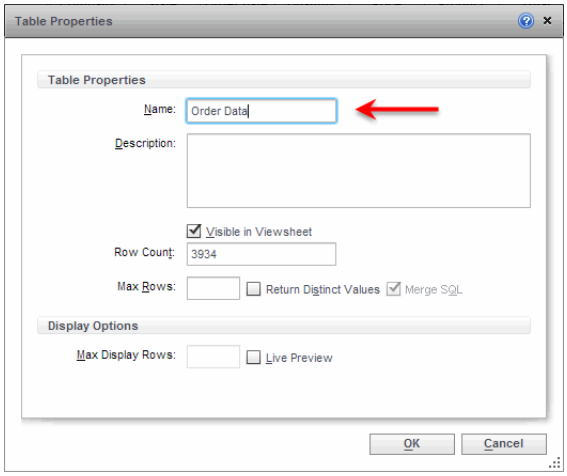
Query1 (Primary)

Company	State	Order Date	Quantity	Price	Product	Total
Interstate Sho	NJ	2010-04-30 0	6	2850	NetStorage	17100
Interstate Sho	NJ	2010-04-30 0	5	59	Animal World	295
Interstate Sho	NJ	2010-04-30 0	1	550	Mega Icons	550
Interstate Sho	NJ	2010-04-30 0	16	248	WebCalendar	3968
Interstate Sho	NJ	2010-06-08 0	18	310	19 inch LCD	5580
Interstate Sho	NJ	2010-11-22 0	18	145	Sync Me	2610
Interstate Sho	NJ	2010-11-22 0	15	235	Math for Me	3525

If you imported the sample *OrderDataText.xls* Excel file, proceed with the steps below to save the data with the Data Worksheet.

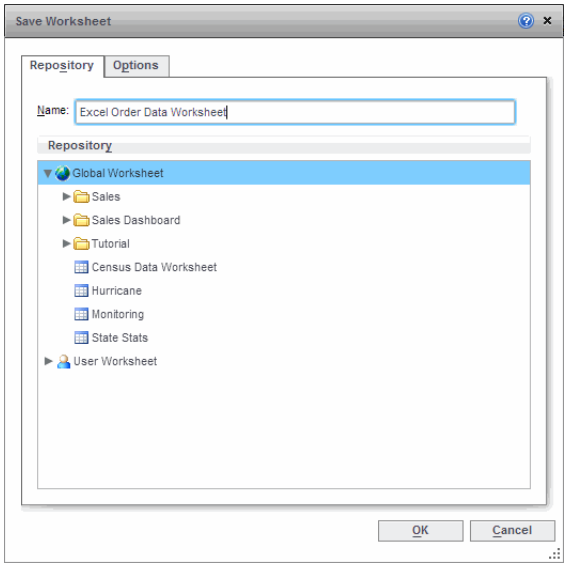
9. Right-click on the table, and select ‘Properties’ from the context menu. This opens the ‘Table Properties’ dialog box.

10. In the ‘Name’ field, enter the label “Order Data”, and press ‘OK’.



11. Press the ‘Save’ button in the toolbar. This opens the ‘Save As’ dialog box.

12. In the ‘Name’ field, enter “Excel Order Data Worksheet”. Select the ‘Global Worksheet’ node on the tree, and press ‘OK’.



This saves the Data Worksheet under the name “Excel Order Data Worksheet” at the root level.

You have now finished importing your data into the Data Worksheet, and you can proceed to create dashboards or reports based on this Data Worksheet.

See Also

Creating a Dashboard, for information on creating a dashboard based on a Data Worksheet.

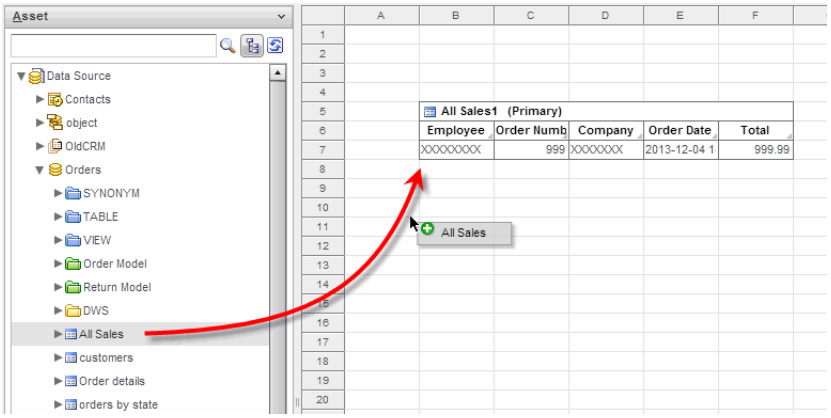
Creating a Report, for information on creating a report based on a Data Worksheet.

Filtering Data

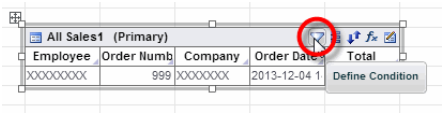
To create filtering conditions on either pre-aggregate or post-aggregate data, follow the steps below. **Note**: If you want dashboard users to dynamically filter the data in a dashboard, see *Adding a Selection (Filter)* instead.

1. Create a new Data Worksheet. See *Creating a Data Worksheet*.

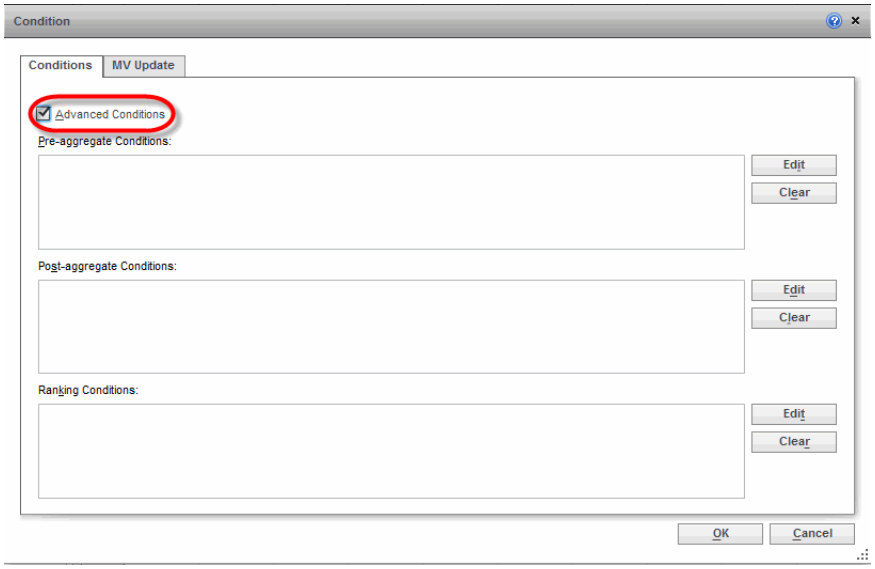
2. From the 'Data Source' listing in the **Asset** panel, drag the query you want to filter into the Worksheet grid. This creates a new data block in the Data Worksheet.



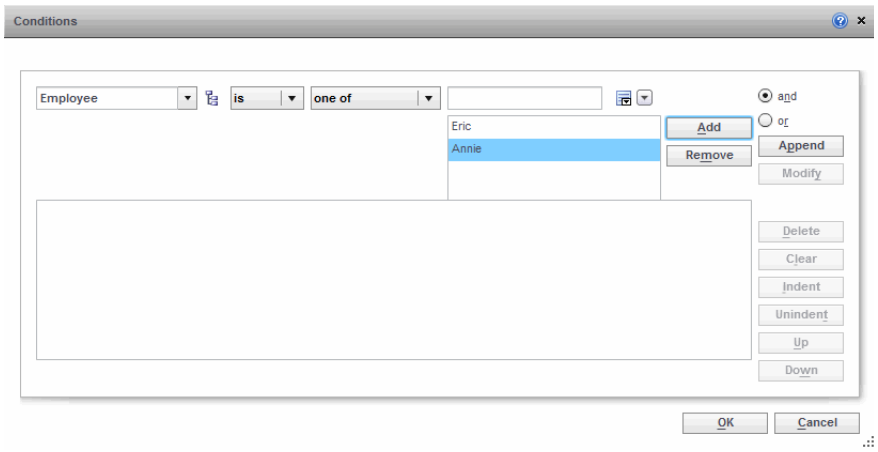
3. Press the 'Define Condition' button on the title bar of a data block. This opens the 'Condition' dialog box.



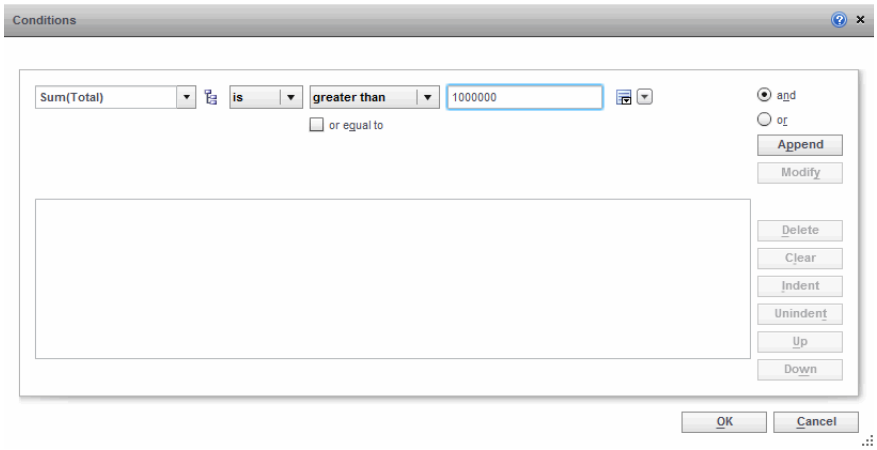
4. Select the 'Advanced Conditions' check box.



5. To create a **pre-aggregate condition** (a condition that applies to detail data prior to grouping and aggregation), press the 'Edit' button next to the 'Pre-aggregate Conditions' panel. Use the menus to specify the desired condition. Then press 'Append' to add the condition and 'OK' to close the dialog box.

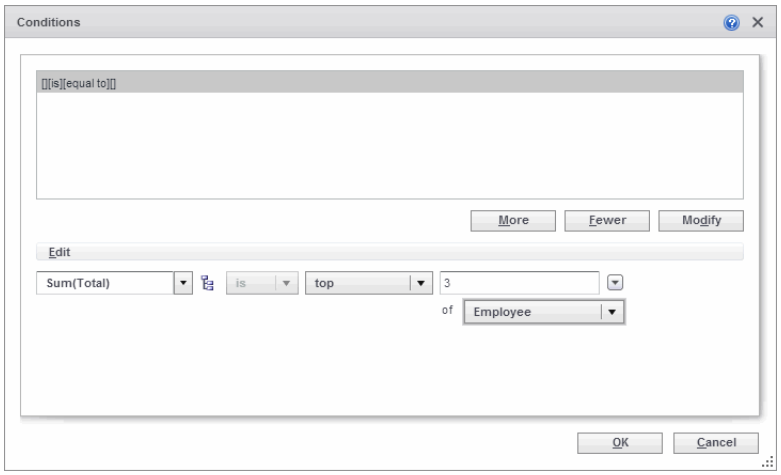


6. To create a **post-aggregate condition** (a condition that applies to summarized data following grouping and aggregation), press the 'Edit' button next to the 'Post-aggregate Conditions' panel. Use the menus to specify the desired condition. Then press 'Append' to add the condition and 'OK' to close the dialog box.



See *Grouping and Aggregating Data* for information on creating aggregate fields.

7. To create a **ranking condition** (a condition that filters the top-*N* or bottom-*N* groups based on a specified measure), press the 'Edit' button next to the 'Ranking Conditions' panel. Press the 'More' button and use the menus to specify the desired condition. Then press 'Append' to add the condition and 'OK' to close the dialog box.



The selected data block is now filtered to return only data matching the specified condition. Save the Data Worksheet by pressing the 'Save' button in the toolbar. You can now use this Data Worksheet as the data source for a dashboard or report. See *Creating a Dashboard* or *Creating a Report*.

See Also

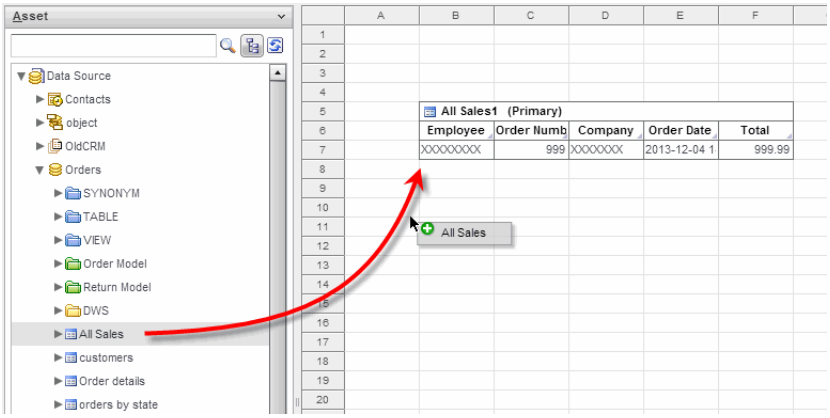
Filtering, in *Advanced Data Mashup* for full information on filtering data blocks.

Grouping and Aggregating Data

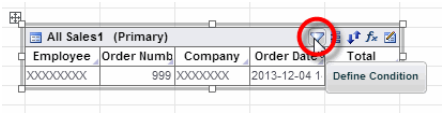
To group and aggregate data, follow the steps below.

1. Create a new Data Worksheet. (See *Creating a Data Worksheet*.)

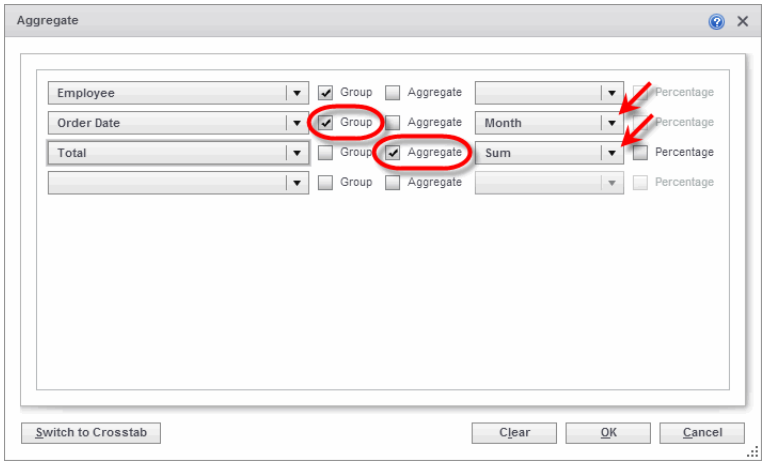
2. From the 'Data Source' listing in the **Asset** panel, drag in the query you want to group and aggregate. This creates a new data block in the Data Worksheet.



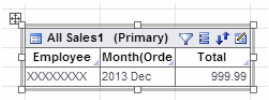
3. Press the 'Group' button on the title bar of a data block. This opens the 'Aggregate' dialog box.



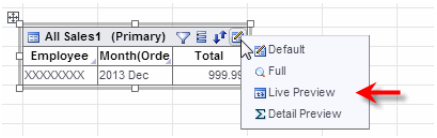
4. Use the menus to select grouping fields and summary fields. For date-type fields, select the desired level of date grouping ('Year', 'Month', etc.). For summary fields, select the desired method of aggregation ('Sum', 'Average', etc.).



5. Press ‘OK’ to close the dialog box. The data block is now grouped, and only the grouping and summary columns are returned in the result set.



6. To view live data, press the ‘Change View’ button in the data block title bar, and choose ‘Live Preview’.



Save the Data Worksheet by pressing the ‘Save’ button in the toolbar. You can now use this Data Worksheet as the data source for a dashboard or report. See *Creating a Dashboard* or *Creating a Report*.

See Also

Grouping, in *Advanced Data Mashup*, for full information on grouping and aggregating data blocks.

Joining Tables

Joins allow you to mash up data from different database tables, queries, data models, or Data Worksheets. A *joined table* is a composition of two tables that preserves a specified relationship between the participating tables (*base tables*).

The joined table contains all the columns from both base tables, but retains only the rows satisfying a specified set of *join conditions*. The join conditions are relations between pairs of columns, one from each base table.

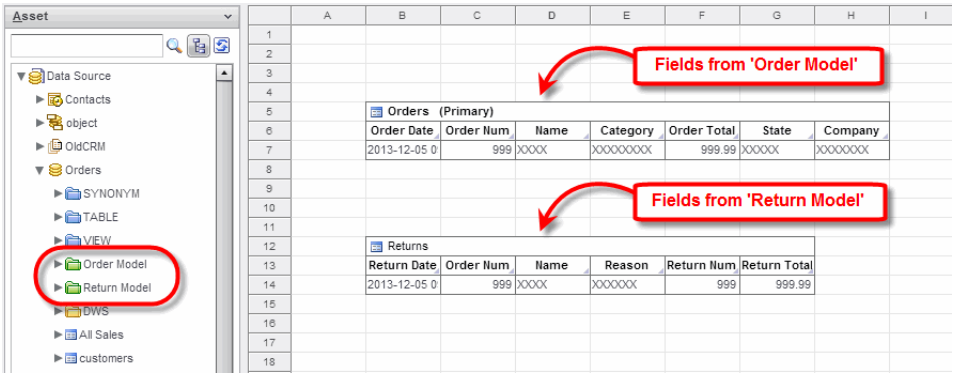
To join tables together using an inner join, outer join, or inequality join, follow the steps below:

1. Create a new Data Worksheet. See *Creating a Data Worksheet*.

2. From the **Asset** panel, drag in the tables, queries, entities, or Data Worksheets you want to join together. Each item that you drag in creates a new data block.

```
A data model is similar to a query, but provides data in
hierarchical form as entities and attributes.
```

In this example, we are joining together two data blocks, one containing data from the sample ‘Return Model’ data model, and the other containing data from the sample ‘Order Model’ data model.

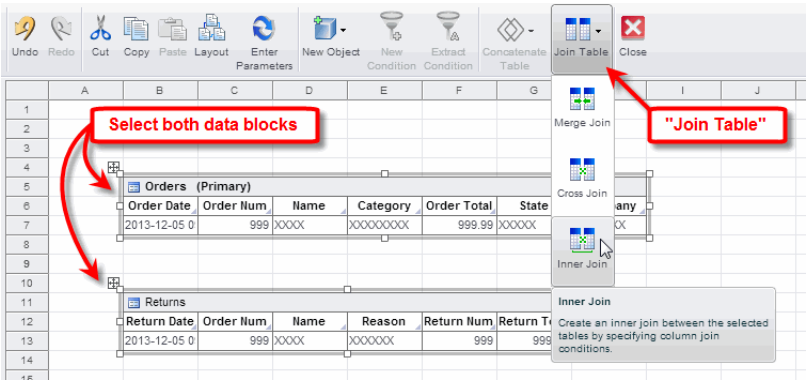


You can do the same mash-up operations with any data blocks, whether they are derived from database tables, queries, data models, or Data Worksheets, and whether they are from the same data source or from different data sources.

3. Select the data blocks you want to join together. (You can Shift-click on the data blocks to select both, or simply drag the mouse across them.)

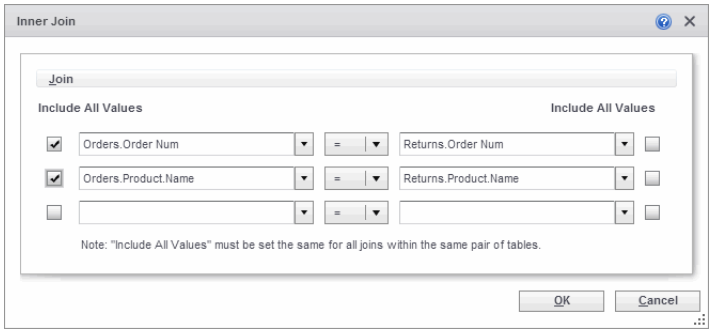
You can also create a join by dragging columns. See *Creating a Inner Join by Dragging Columns* in *Advanced Data Mashup*.

4. Press the 'Join Table' button in the toolbar, and select the 'Inner Join' option.



This opens the 'Inner Join' dialog box.

5. Specify the join conditions. Each join condition specifies how a pair of columns from different data blocks should be related.



For example, the first join condition specifies that the values in the 'Order Num' column from the 'Orders' data block should be matched to the values in the 'Order Num' column from the 'Returns' data block.

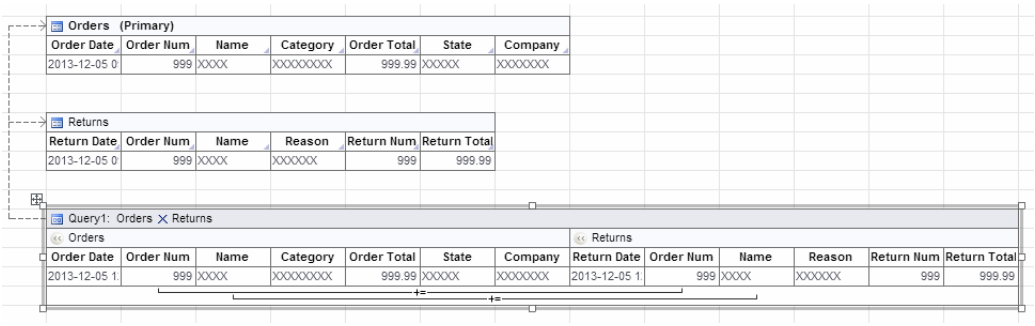
```
Orders.Order Num = Returns.Order Num
```

The second join condition specifies that the values in the 'Name' column from the 'Orders' data block should be matched to the values in the 'Name' column from the 'Returns' data block

```
Orders.Product.Name = Returns.Product.Name
```

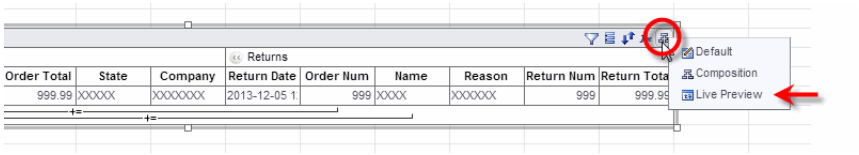
6. Select the ‘Include All Values’ option next to the left-side ‘Orders’ data block fields. This creates a *left outer join*, which includes all values from the ‘Orders’ table in the result, even if these orders have no matches in the ‘Returns’ data block.

7. Press ‘OK’ to create the joined data block.

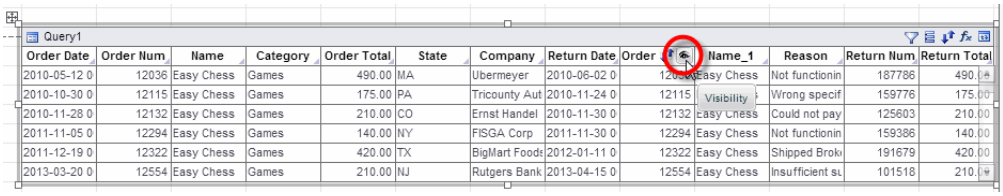


The resulting data block is displayed in ‘Composition View’, which shows the original ‘Orders’ and ‘Returns’ tables and how they are joined.

8. To view the actual data, press the ‘Change View’ button in the data block title bar and select ‘Live Preview’.

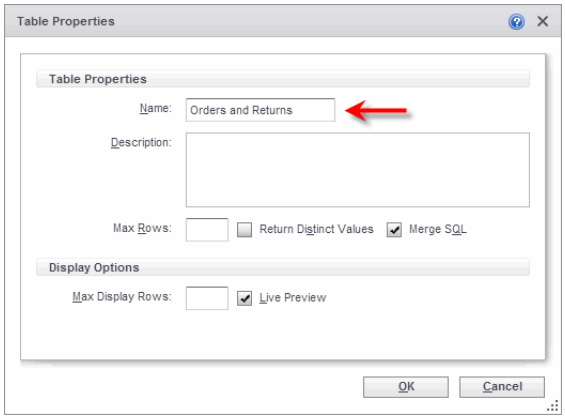


9. To remove duplicate columns (e.g., the join columns), press the ‘Visibility’ icon in the column header.

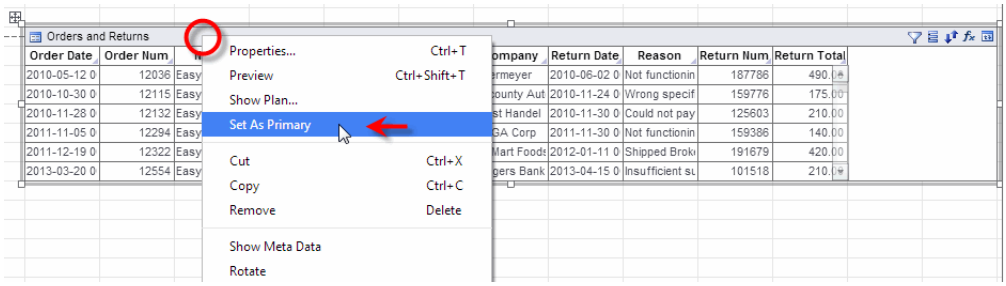


10. To change column names, double click the column header and edit the text.

11. To rename the joined data block, right-click the title bar and select ‘Properties’ from the context menu. This opens the ‘Table Properties’ dialog box. Enter the desired name in the ‘Name’ field, and press ‘OK’.



12. Right-click the joined data block, and select ‘Set As Primary’ from the context menu.



It is good practice to mark the final result of your Data Worksheet as the **primary** data block. This ensures that this data block will be available to any report based on this Data Worksheet, and will be listed first among the data blocks available to any dashboard based on this Data Worksheet.

Save the Data Worksheet by pressing the ‘Save’ button in the toolbar. See *Saving a Data Worksheet*. You can now use this Data Worksheet as the data source for a dashboard or report. See *Creating a Dashboard* or *Creating a Report*.

See Also

Joining Tables, in *Advanced Data Mashup*, for full information about joins.
What-If Analysis, in *Advanced Data Mashup*, for an application of joins.
Changing the Join Type, in *Advanced Data Mashup*, for information on converting between join types.
Editing a Composition Table, in *Advanced Data Mashup*, for information on how to use the editable and composition views of tables.

Concatenating Tables

Union, intersection, and difference operations are known as table *concatenations*. Tables must have the same number of columns in order to participate in concatenations, and corresponding columns of each table must have the same data type.

To concatenate multiple tables, follow these steps:

```
The order in which you select the tables will influence the result
for a ‘Minus’ operation.
```

1. Select the tables to concatenate. (Ctrl-click the table title rows to select them, or drag across the tables.)



2. Press the ‘Concatenate Table’ button in the Worksheet toolbar and select the desired concatenation: ‘Union’, ‘Intersect’, or ‘Minus’. See *Union*, *Intersect*, and *Minus* in *Advanced Data Mashup* for more details.

See *Changing the Concatenation Type* in *Advanced Data Mashup* for information about switching the concatenation type of an existing concatenated table.

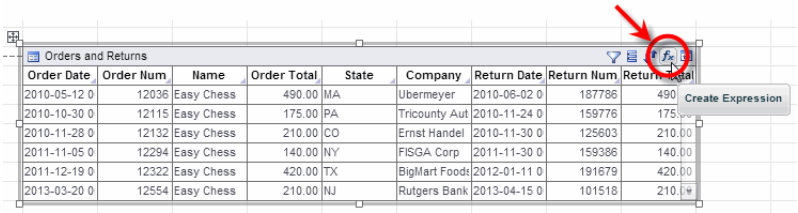
Creating an Expression

An *expression column* is a table column that you define using either JavaScript or SQL. The expression is evaluated for each row of the data block.

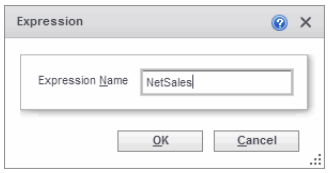
To add an expression column to a data block, follow the steps below:

1. Create a Data Worksheet. (See *Creating a Data Worksheet*.)

2. Press the 'Expression' (*fx*) button in the title bar of the data block to which you want to add an expression column. This opens the 'Input' dialog box.



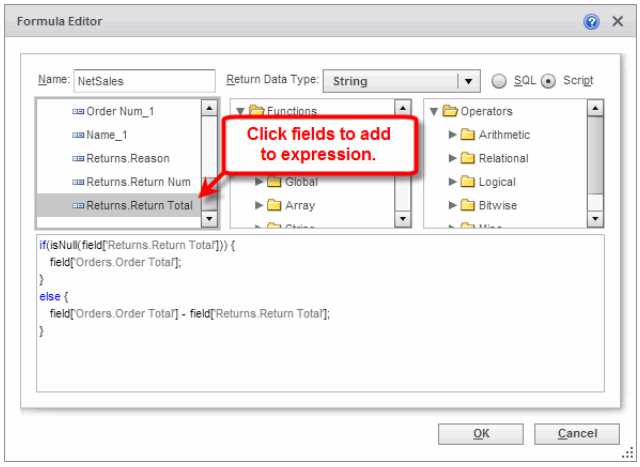
3. In the 'Input' dialog box, enter the name of the new column. Press 'OK' to open the Expression Editor.



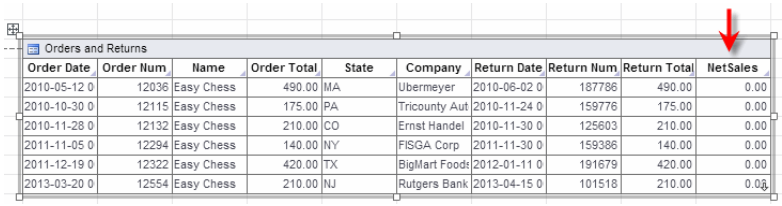
4. At the top of the Expression Editor, select either 'SQL' or 'Script'. Select 'SQL' (if available) to write the expression using your database's SQL syntax. Select 'Script' to write the expression using JavaScript.

5. From the 'Return Data Type' menu, select the appropriate data type for the values that the expression will return.

6. Type the desired expression into the script area. To add a field into the expression, click on the field name in the 'Fields' listing in the left panel.



7. Press ‘OK’ to close the Editor and create the new column.



Orders and Returns

Order Date	Order Num	Name	Order Total	State	Company	Return Date	Return Num	Return Total	NetSales
2010-05-12 0	12036	Easy Chess	490.00	MA	Ubermeyer	2010-06-02 0	187786	490.00	0.00
2010-10-30 0	12115	Easy Chess	175.00	PA	Tricounty Aut	2010-11-24 0	159776	175.00	0.00
2010-11-28 0	12132	Easy Chess	210.00	CO	Ernst Handel	2010-11-30 0	125603	210.00	0.00
2011-11-05 0	12294	Easy Chess	140.00	NY	FISGA Corp	2011-11-30 0	159386	140.00	0.00
2011-12-19 0	12322	Easy Chess	420.00	TX	BigMart Foods	2012-01-11 0	191679	420.00	0.00
2013-03-20 0	12554	Easy Chess	210.00	NJ	Rutgers Bank	2013-04-15 0	101518	210.00	0.00

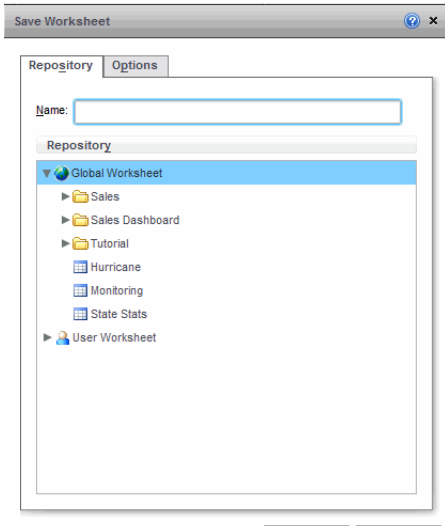
Saving a Data Worksheet

To save a Data Worksheet, follow these steps:

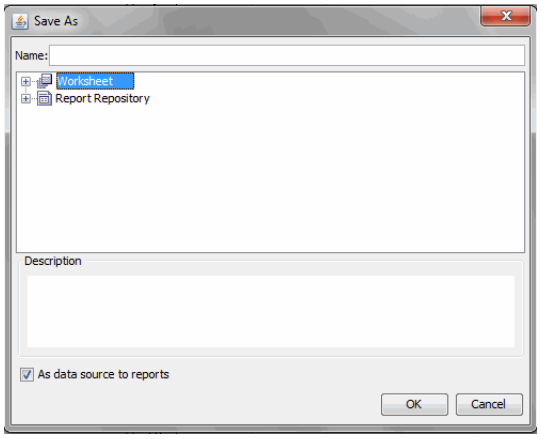


1. Press the ‘Save’ button in the toolbar, or press Ctrl-S on the keyboard. This opens the ‘Save Worksheet’ dialog box.

   In Visual Composer, the ‘Save Worksheet’ dialog box appears as shown below:



   In Style Studio, the ‘Save Worksheet’ dialog box appears as shown below:



2. Enter a name for the Data Worksheet in the ‘Name’ field.

3. Select a location in which to save the Data Worksheet.

4. (Optional) If you are using Visual Composer, under the **Options** tab, set the ‘Design mode sample data size’ to the maximum number of rows that you want tables in the Data Worksheet to retrieve.

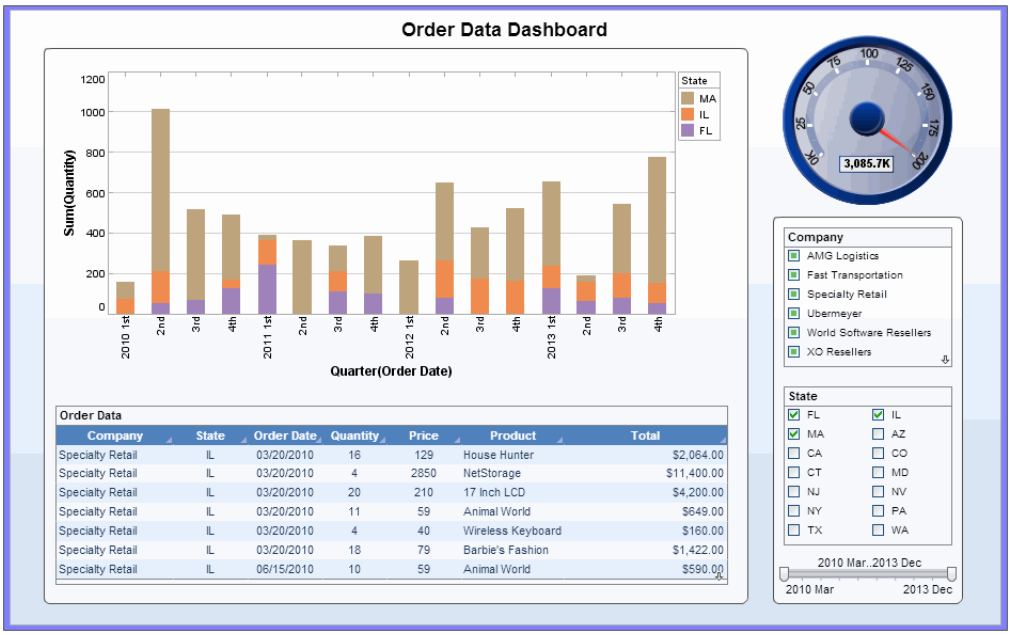
5. Press ‘OK’ to save the Data Worksheet.

Data Worksheets are stored in a file called *asset.dat* in the working repository.

Creating a Dashboard

To build a dashboard, lay out components such as gauges, tables, and charts on a simple canvas called a Viewsheet. A Viewsheet can retrieve data from a query, data model, or Data Worksheet.

The following sections explain how to build a dashboard such as the one shown below.



Creating a New Dashboard

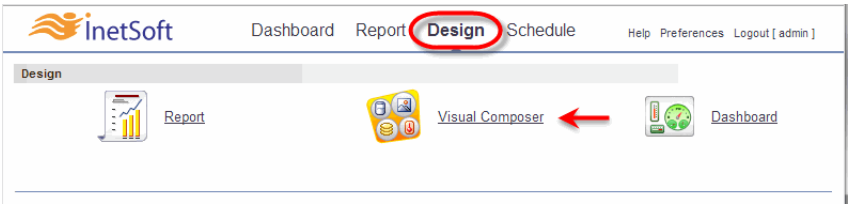
To create a new dashboard (also called a Viewsheet) follow the steps below:

See *Using the Portal* for more information about features of the User Portal.

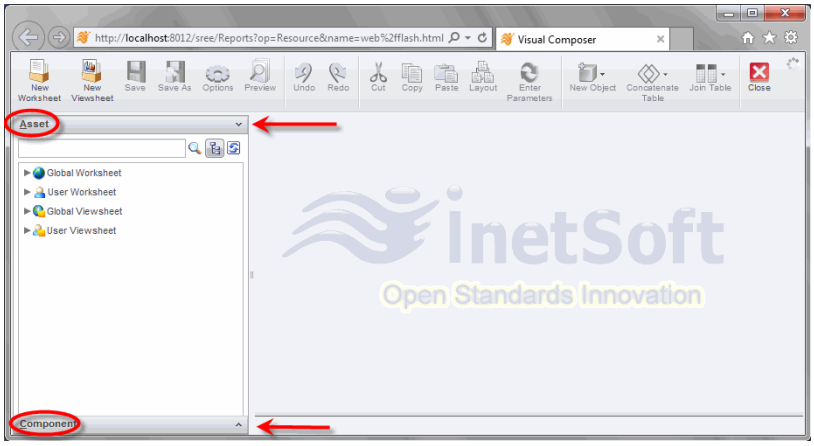


1. Log into the User Portal. (See *Logging into the Portal*.)

2. In the User Portal, select the **Design** tab press the 'Visual Composer' link.



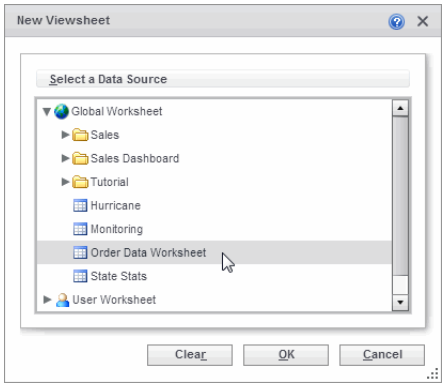
This opens Visual Composer in a new window. The left-side panels in Visual Composer are labeled **Asset** and **Component**. The **Asset** panel lists all Data Worksheets and Viewsheets. The **Component** panel lists all of the components available to a Viewsheet.





3. Press the 'New Viewsheet' button. This opens the 'New Viewsheet' dialog box.

4. In the 'New Viewsheet' dialog box, select the desired data source (Data Worksheet, query, data model, or database table). Press 'OK'.



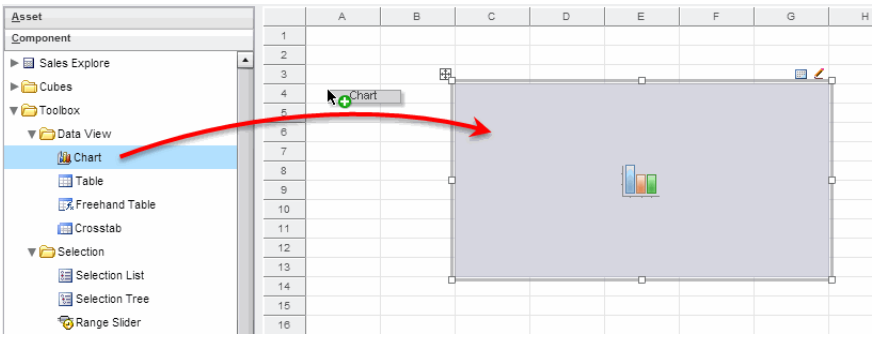
This creates a new dashboard (Viewsheet) that draws its data from the selected data source.

Adding a Chart

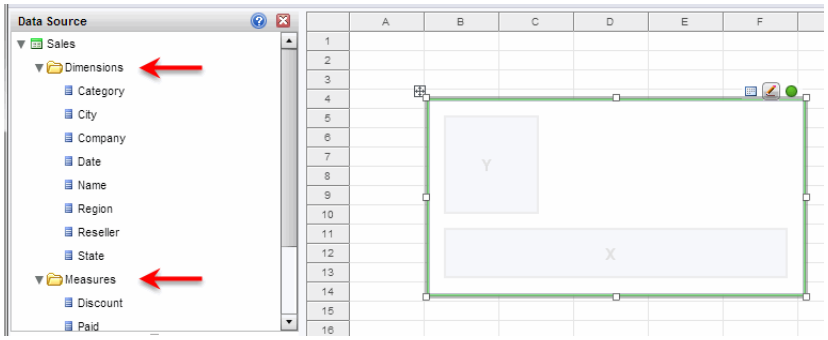
To create a chart, follow the steps below:

1. If necessary, create a new Viewsheet. See *Creating a New Dashboard*.

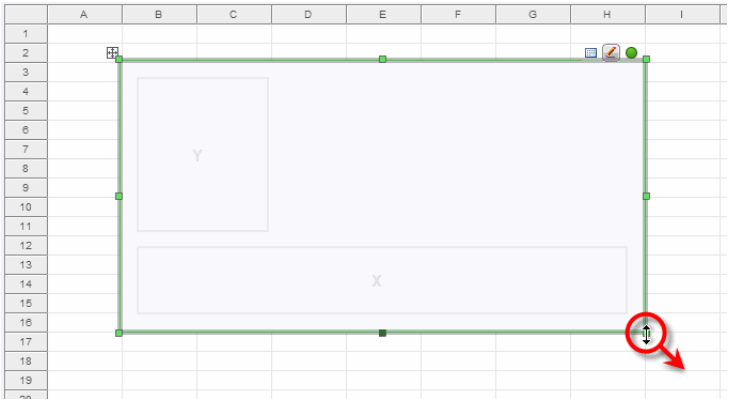
2. From the **Component** panel, drag a Chart component into the Viewsheet grid. This adds the Chart to the Viewsheet and opens the Chart Editor in the left panel.



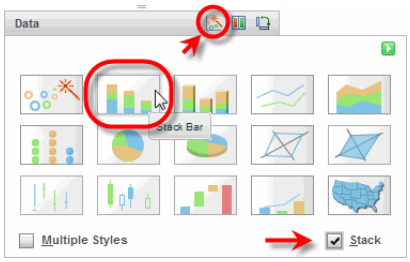
A chart visualizes a *dataset*, which is a table containing rows and columns. Columns (also called *fields*) in the dataset are classified as either 'Dimensions' or 'Measures' in the Chart Editor.



3. Resize the chart as desired by dragging the handles.



4. (Optional) Set the style of the chart. To do this, press the ‘Select Chart Style’ button in the **Data** panel. Choose a desired style, such as ‘Line’ or ‘Bar’, and press the green ‘Apply’ button.

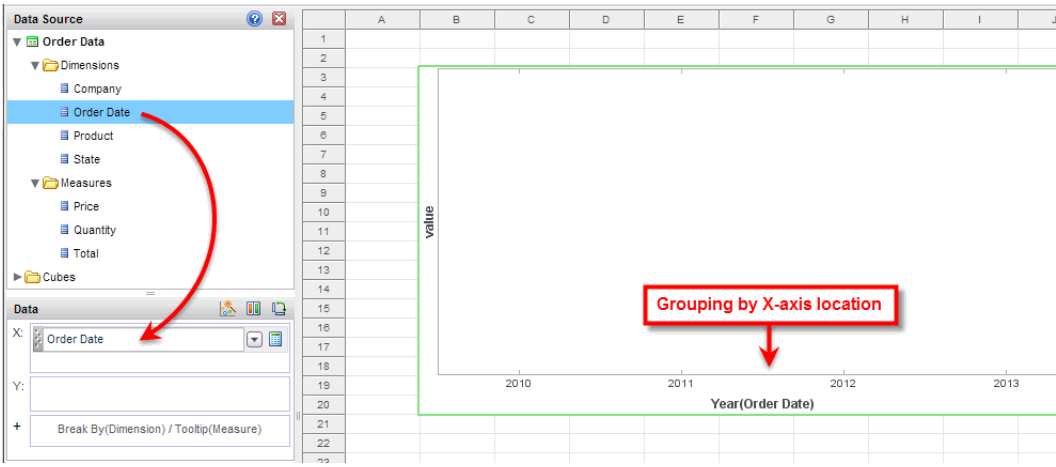


`Note: If you select a ‘Pie’ style, you should place the dimension field on either the ‘Color’ or ‘Shape’ region of the` **`Visual`** `panel.`

To create a stacked chart, enable the ‘Stack’ option at the bottom of the panel. If you do not select a specific chart style, the default ‘Auto’ style automatically selects a chart style appropriate to your dataset.

To convert a measure to a dimension, right-click the measure in the **Data Source** panel and select ‘Convert to Dimension’.

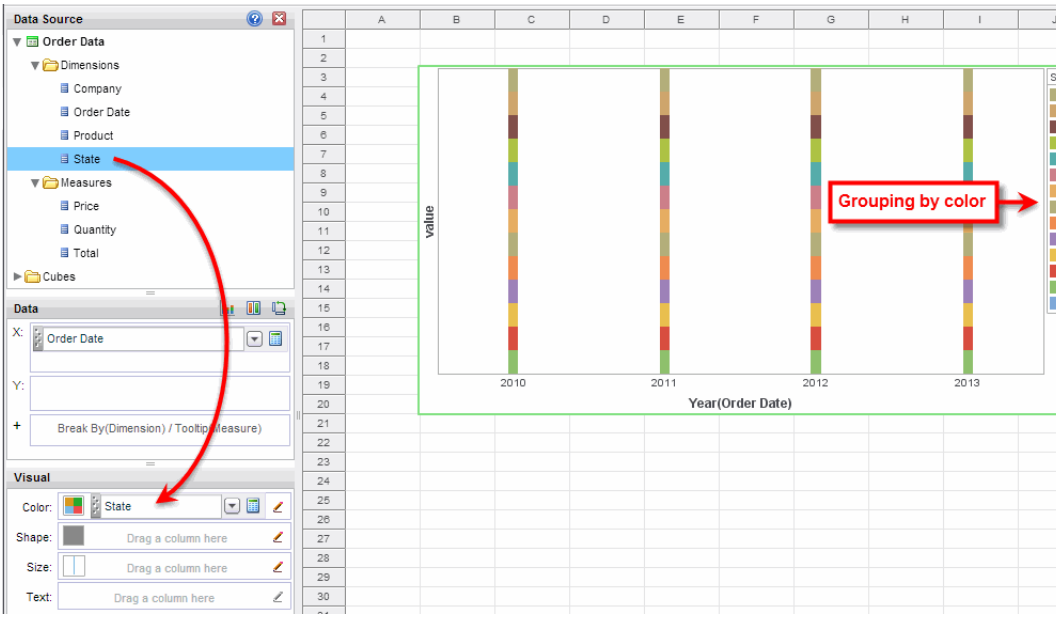
5. From the ‘Dimensions’ folder of the **Data Source** panel, drag a desired dimension to the ‘X’ region.



Dimensions are used to break-down the dataset into multiple groups. Adding a dimension to the 'X' region distinguishes the different dimension groups by location on the X-axis. Adding a dimension to the 'Y' region distinguishes the different dimension groups by location on the Y-axis. You can add multiple dimensions into the 'X' or 'Y' regions to create multiple grouping levels.

You can also distinguish groups in a dimension by using color, shape, size, or label, as described in the next step.

6. To break-out the dataset into groups using color, shape, size, or text labeling, drag a dimension from the **Data Source** panel to one of the regions in the **Visual** panel.



a. To denote different groups using different colors, drag the field to the ‘Color’ region.

b. To denote different groups using different shapes, drag the field to the ‘Shape’ region.

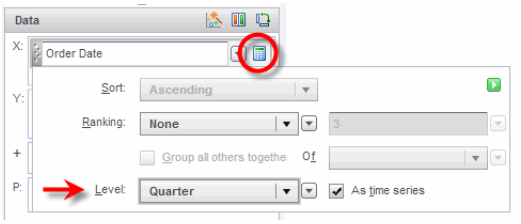
c. To denote different groups using different sizes, drag the field to the ‘Size’ region.

d. To denote different groups using different text labels, drag the field to the ‘Text’ region.

Note that the effect of color, shape, and size codings varies by chart style. For example, in a bar-type chart the ‘Shape’ representation affects the style of the fill pattern, whereas in a point-type chart the ‘Shape’ representation affects the image of the point.

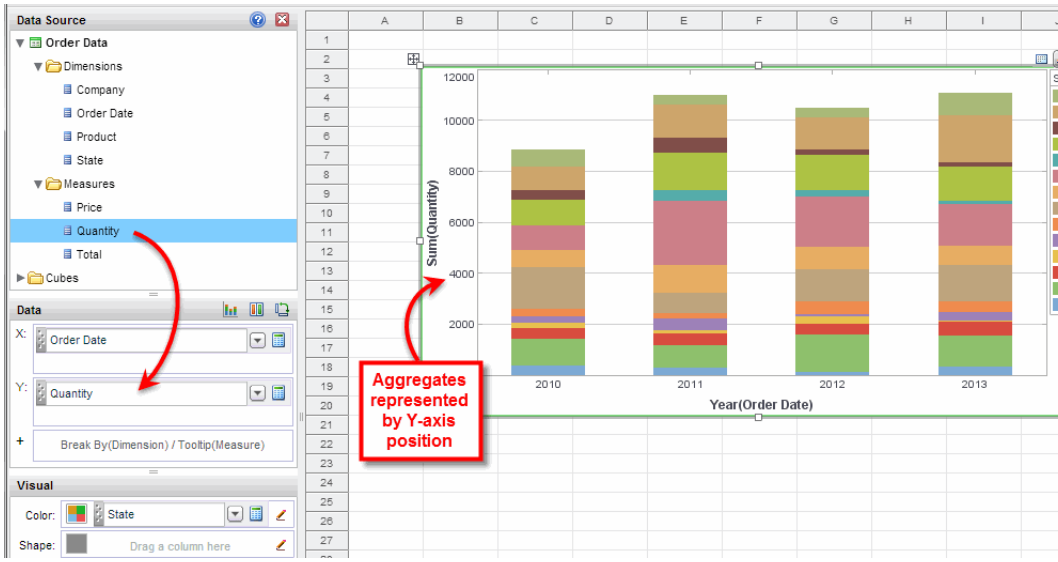
7. To break-out the data into groups *without* applying any visual formatting, drag a dimension to the ‘Break By/Tooltip’ region.

8. (Optional) For a date dimension, press the ‘Edit Dimension’ button next to the field name in the Chart Editor, and set the ‘Level’ to the desired date grouping. Then press the green ‘Apply’ button.



You can add multiple measures to a chart. To convert a dimension to a measure, right-click the dimension in the **Data Source** panel and select ‘Convert to Measure’.

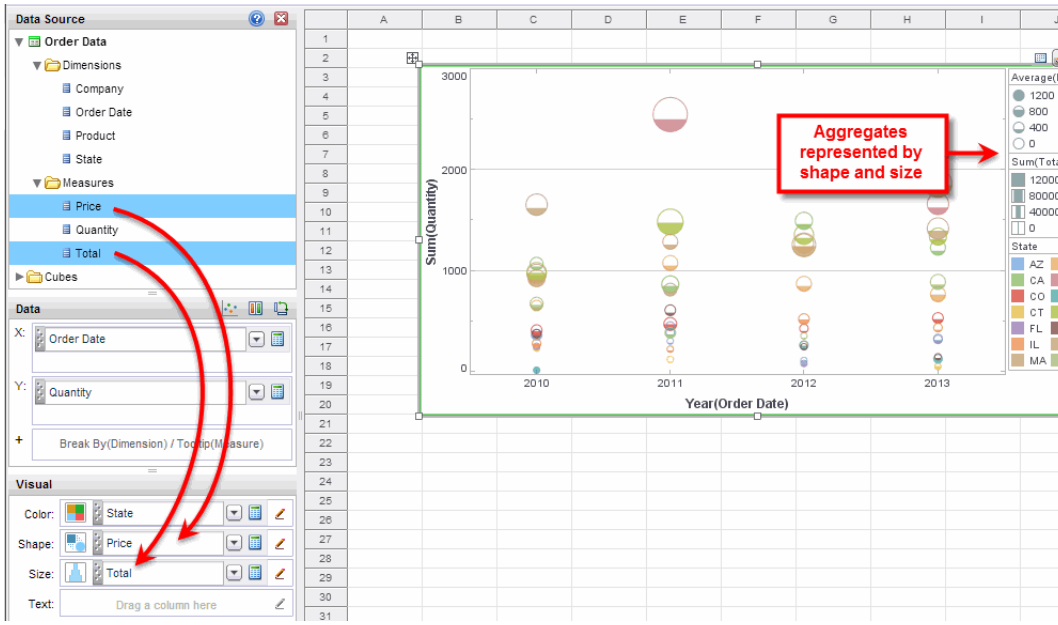
9. From the ‘Measures’ folder of the **Data Source** panel, drag a measure to the ‘Y’ region. This places the selected field onto the chart as a measure.



A measure is generally used for aggregation, for example summation, averaging, correlation, etc. Adding a measure to the 'Y' region displays the computed aggregates by using locations on the Y-axis. Adding a measure to the 'X' region displays the computed aggregates by using locations on the X-axis.

You can also display aggregates by using color, shape, size, or label, as described in the next step.

10. To represent a measure using color, shape, size, or labeling, drag the desired measure field from the **Data Source** panel to a region of the **Visual** panel.



a. To represent the measure value using a continuous color spectrum, drag the measure to the ‘Color’ region.

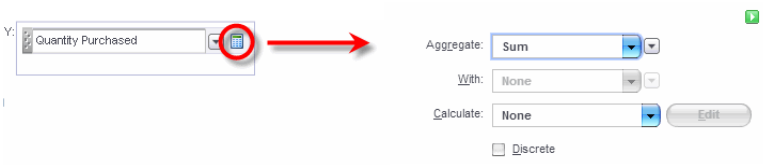
b. To represent the measure value using a continuous shape spectrum, drag the measure to the ‘Shape’ region.

c. To represent the measure value using a continuous size spectrum, drag the measure to the ‘Size’ region.

d. To represent the measure by placing numerical values on the chart, drag the measure to the ‘Text’ region.

Note that the effect of color, shape, and size codings varies by chart style. For example, in a bar-type chart the ‘Shape’ representation affects the density of the fill pattern, whereas in a point-type chart the ‘Shape’ representation affects the geometry of the point or the fraction filled.

11. Press the ‘Edit Measure’ button next to the measure, and select the desired aggregation method for the measure.



12. To represent a measure by using tooltip information, drag the measure to the ‘Break By/Tooltip’ region.

For full information on creating charts, see *Charts* in *Advanced Dashboard Design*.

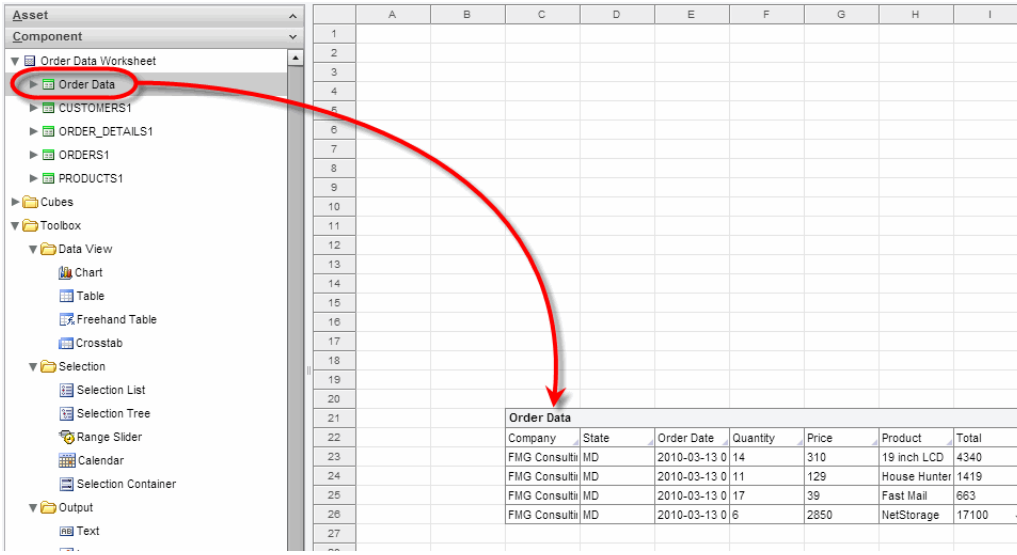
Adding a Table

To create and format a table, follow the steps below:

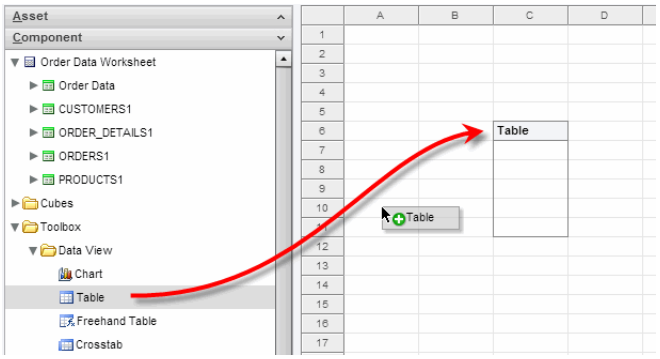
1. If necessary, create a new Viewsheet. See *Creating a New Dashboard*.

2. Add a new table using one of the following approaches:

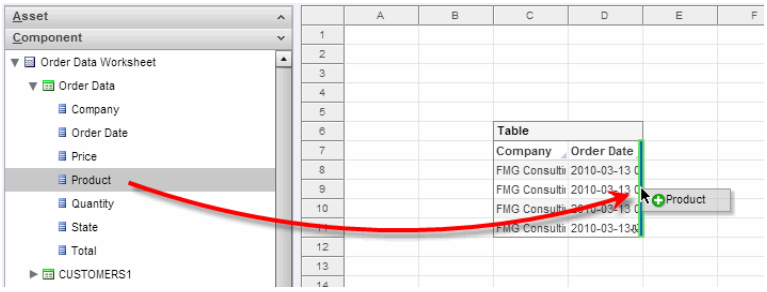
   a. Expand the data source node at the top of the **Component** panel in Visual Composer. Drag an entire data block into the Viewsheet grid. This adds the data block to the Viewsheet as a Table component.



   b. Alternatively, drag a Table component from **Component** panel into the Viewsheet grid to create a blank table.

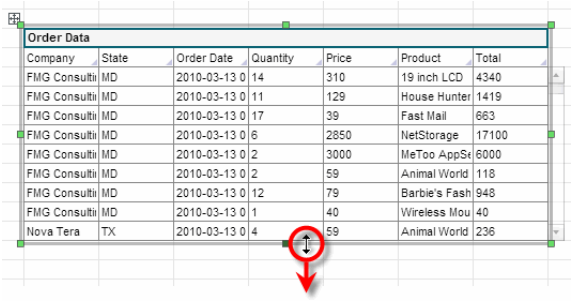


Then drag individual columns from a data block into the table. The green highlight shows you where each column will be placed.

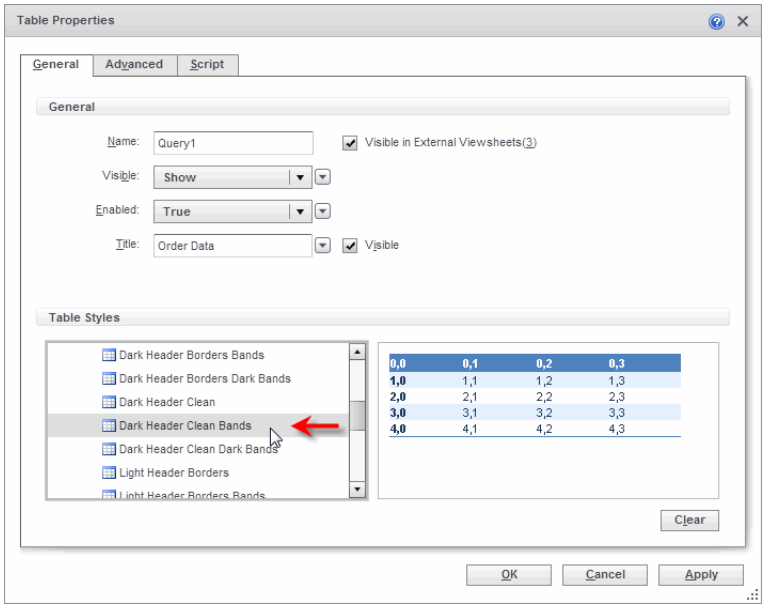


Proceed with steps below to format the table as desired.

3. Click the table to select it. Drag the bottom border of the table to display more rows.

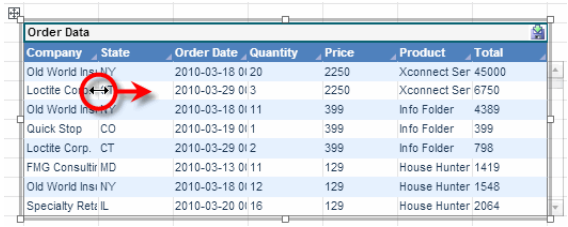


4. Right-click the table and select ‘Properties’ from the context menu. This opens the ‘Table Properties’ dialog box. In the ‘Table Styles’ panel of the **General** tab, select a table style. Press ‘OK’.

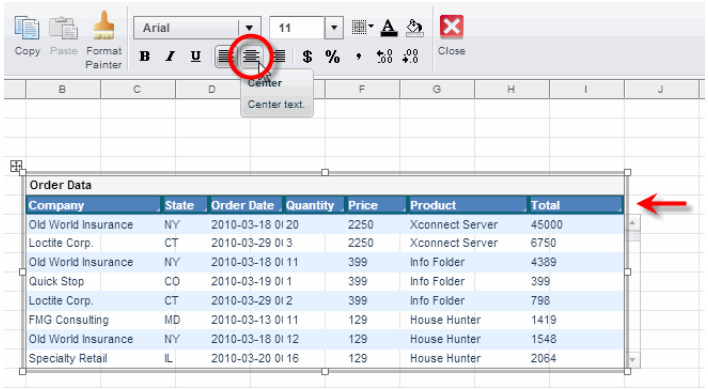


This applies a global style to the table, including fonts, background colors, foreground colors, and borders.

5. Drag the column borders as necessary to resize the columns.

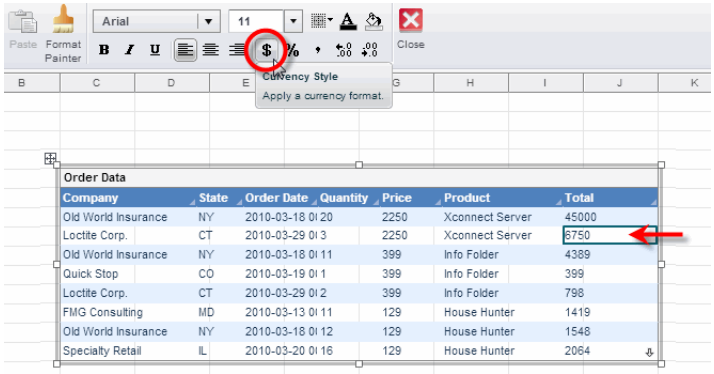


6. To align a column header, click the column header to select it (Shift-click or Ctrl-click to select multiple columns). Then press the desired alignment button in the toolbar to align the header text.



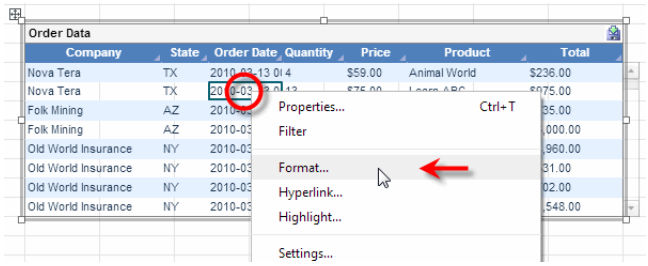
7. To align the data values in a column, select any cell in the column and press the desired alignment button in the toolbar to align the text.

8. To format a column's data as currency, select any cell in the column and press the 'Currency Style ($)' button in the toolbar. This formats the column as currency.

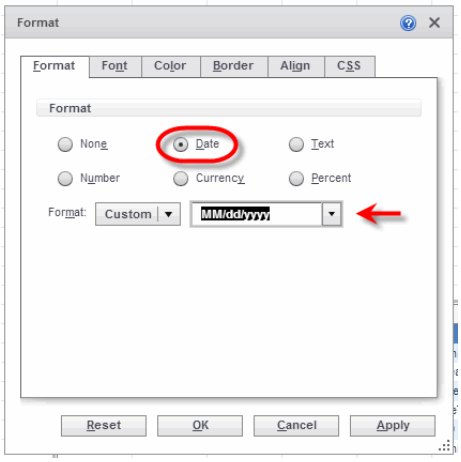


See *Format Dialog Box* in *Advanced Dashboard Design* for more information about formatting.

9. To format a column's data as dates, right-click on any data cell in the column and select 'Format' from the context menu. This opens the 'Format' dialog box.

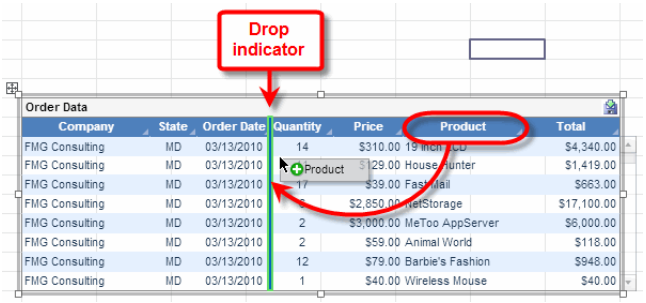


10. In the 'Format' dialog box, under the **Format** tab, select the 'Date' option. Select a date code from the menu or enter your own. (For example, the code `MM/dd/yyyy` indicates that dates should be displayed with a two-digit month, two-digit day, and four-digit year.)



Press ‘OK’ to close the dialog box.

11. To change the order of the columns, click and drag on a column header and drop the column into the desired position. A green highlight indicates the drop position.



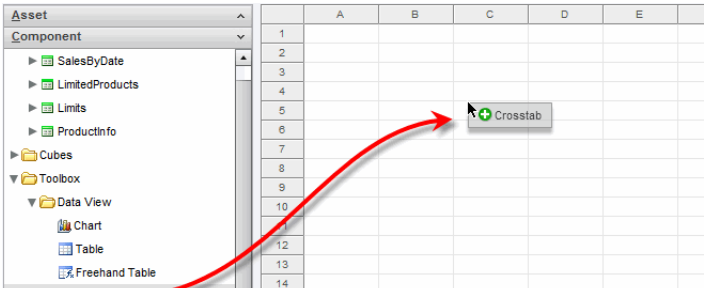
For complete information about creating tables, see *Tables* in *Advanced Dashboard Design*.

Adding a Crosstab

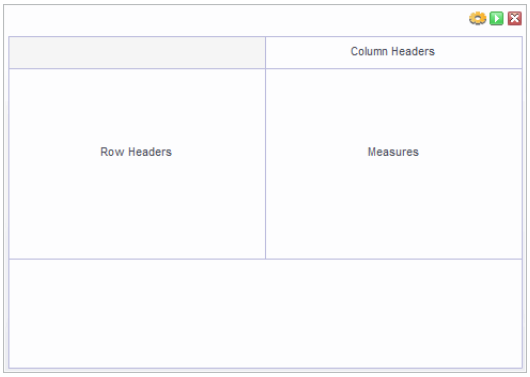
To add a crosstab to a dashboard, follow the steps below:

1. If necessary, create a new Viewsheet. See *Creating a New Dashboard*.

2. Drag the Crosstab element from the **Component** panel into the Viewsheet grid.



This automatically opens the Crosstab Editor.



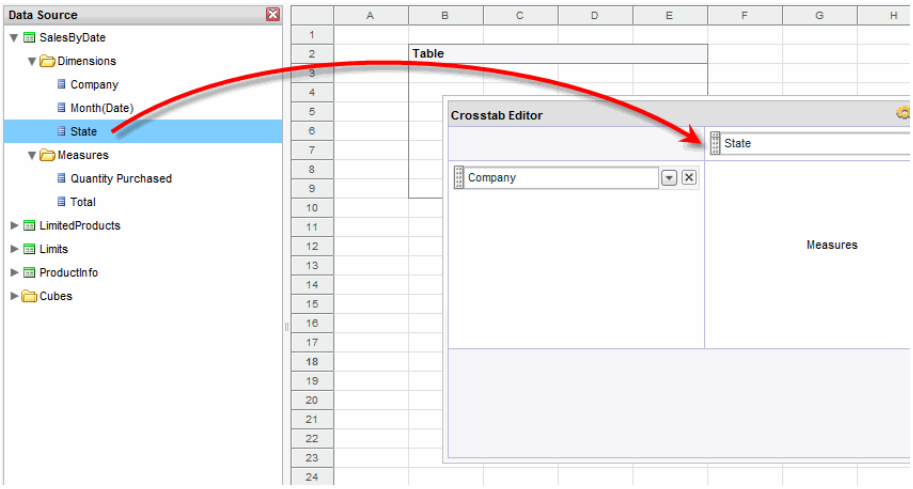
3. If the Editor is not open, move the mouse over the top right corner of the crosstab, and press the 'Edit' button to open the Editor.



The panel displays all OLAP cubes defined on the data source.

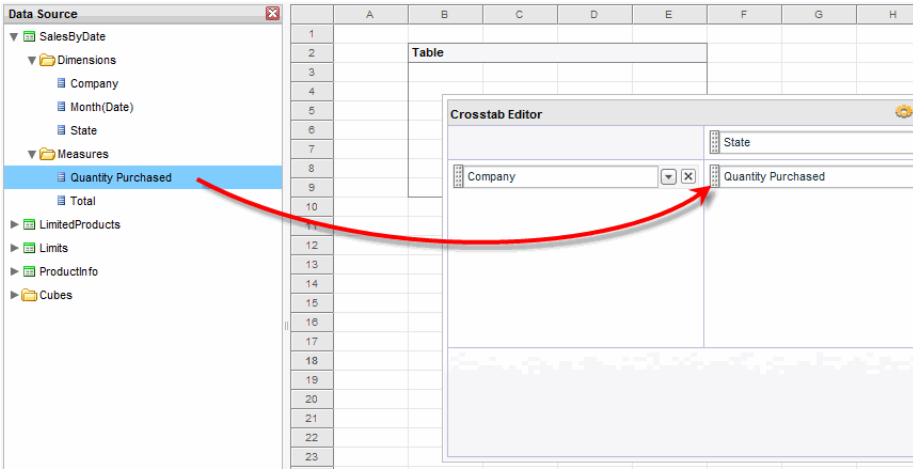
4. In the **Data Source** panel, expand a data block, query, data model, or OLAP cube (hierarchical model) to display the available dimensions and measures.

5. Drag the desired columns or cube dimensions into the ‘Column Headers’ and ‘Row Headers’ regions.



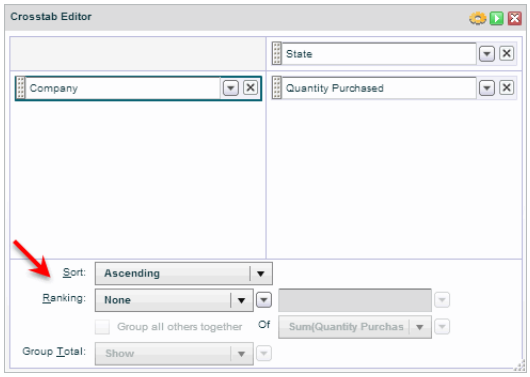
You can change the order of multiple dimensions or measures by dragging them up or down in the Editor.

6. Drag the desired columns or cube measures into the region labeled ‘Measures’.



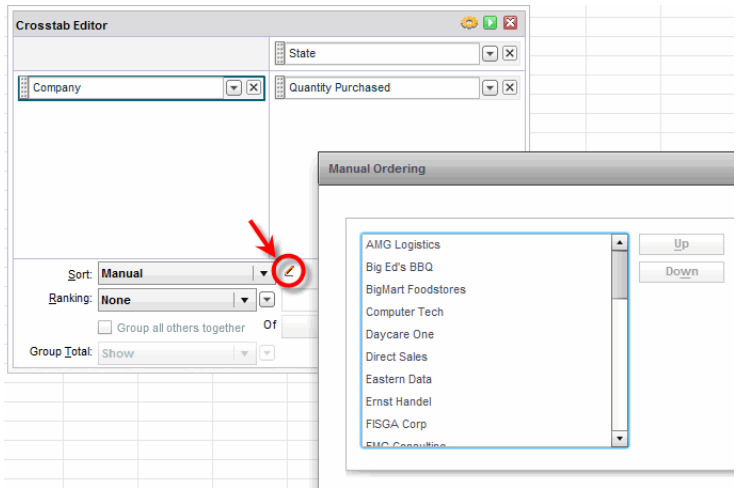
7. (Optional) Specify group totals and ranking:

   a. Click on a row header or column header. This activates the ‘Sort’, ‘Ranking’, and ‘Group Total’ controls.



b. In the ‘Sort’ menu, specify how the values in the field should be ordered. ‘Ascending’ and ‘Descending’ sort the values in alphabetical or reverse-alphabetical order. ‘By Value (Asc)’ and ‘By Value (Desc)’ sort the values according to the aggregate measure selected in the adjacent menu.

   To specify an arbitrary sort order, select the ‘Manual’ option in the ‘Sort’ menu, and press the ‘Edit’ button to open the ‘Manual Ordering’ dialog box. To reorder an item in the ‘Manual Ordering’ dialog box, drag the item up or down within the list or select the item and press the ‘Up’ or ‘Down’ buttons. Press ‘OK’ to close the dialog box.

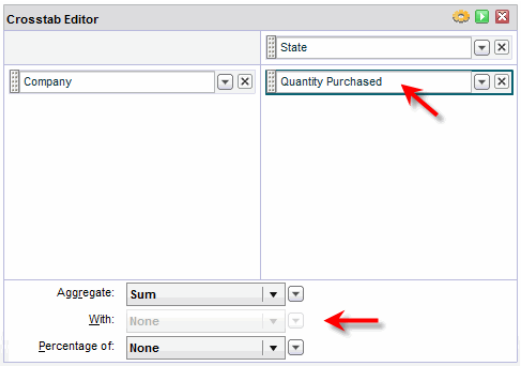


c. In the ‘Ranking’ menu, select the ‘Top’ or ‘Bottom’ filter, and specify the (integer) number of top or bottom groups to select. In the adjacent menu, specify the measure by which groups should be ranked.

   Select ‘Group all others together’ to create an “Others” category which represents (in aggregate) all the groups excluded by the Top/Bottom ranking filter.

d. In the ‘Group Total’ menu, specify whether group totals should be shown.

8. Click on a measure, and select the summarization method from the ‘Aggregate’ menu.

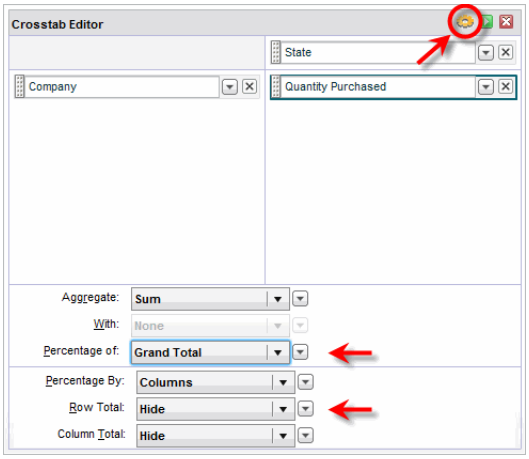


To display a bivariate aggregation measure (e.g., ‘Correlation’, ‘Weighted Average’, etc.), select the second operand (column) from the ‘with’ menu.

9. (Optional) Set any desired percentage options:

a. To display a univariate measure (‘Sum’, ‘Count’, etc.) as a percentage, select the basis for percentage in the ‘Percentage of’ menu.

b. If you choose a percentage representation, click the ‘Options’ button (at the top of the dialog box) to select the dimension along which the percentage is computed (‘Rows’ or ‘Columns’). Percentage by rows uses the row totals, and by columns uses the column totals.



10. Press the green ‘Apply’ button to close the Crosstab Editor.

11. If desired, expand the crosstab horizontally so that all of the column headers are visible.

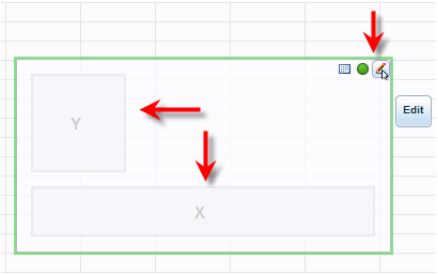
For complete information about creating crosstabs, see *Crosstabs* in *Advanced Dashboard Design*.

Adding a Map

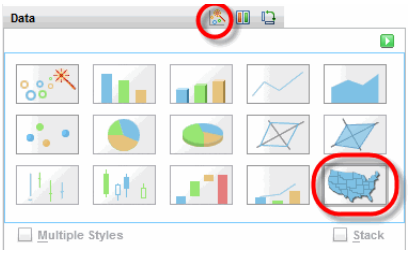
To add a map chart to a Viewsheet, first drag the Chart component from the **Component** panel into the grid. (See *Adding a Chart*.)

To bind the chart to geographical data, follow the steps below:

1. Click on the center of the chart, or click the 'Edit' button in the top-right corner. This opens the Chart Editor on the left side of the Visual Composer.



2. In the Chart Editor's **Data** panel, press the 'Select Chart Style' button, and choose the 'Map' chart type. Press the 'Apply' button (or double-click the 'Map' icon).

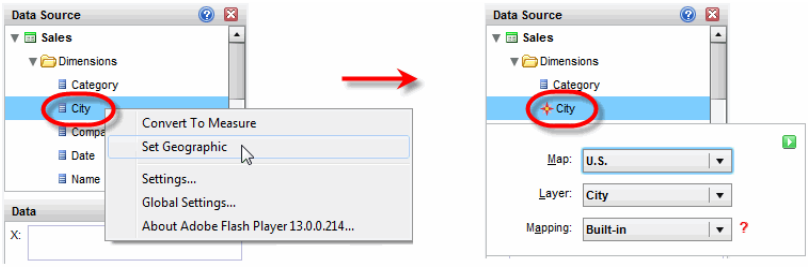


The **Data** panel now provides a 'Geographic' field.

If your columns contain longitude and latitude data (in decimal format) rather than location names, use the 'Longitude' and 'Latitude' fields of the 'Data' panel.

3. Mark the dimensions you want to use on the map as *geographical* fields. To mark a field as geographical, follow the steps below:

   a. Right-click the field in the tree, and select 'Set Geographic' from the context menu. This places a "compass" icon next to the field name, and opens the geographical level editor.



The map that you select for a geographic field applies to all other geographical fields.

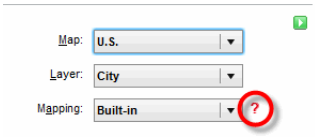
b. In the level editor, select the desired region from the 'Map' menu: Asia, Canada, Europe, Mexico, U.S., or World.

c. From the 'Layer' menu, select the geographical layer that corresponds to the data in the selected field.

   This specifies the layer of the geographical database against which field values should be matched (e.g., city names should be matched against the city layer, etc.). The options available in the 'Layer' menu depend on the previously selected 'Map' option.

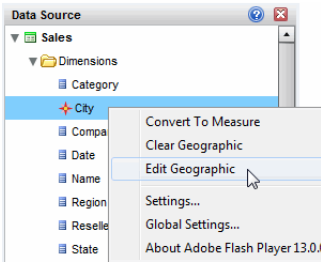
d. From the 'Mapping' menu, select the desired map database.

   The Chart Editor attempts to resolve values in the selected geographic field against the corresponding layer in the map database. If all data values are successfully matched, a green check mark is displayed. If some data values cannot be matched, a red question mark is displayed.



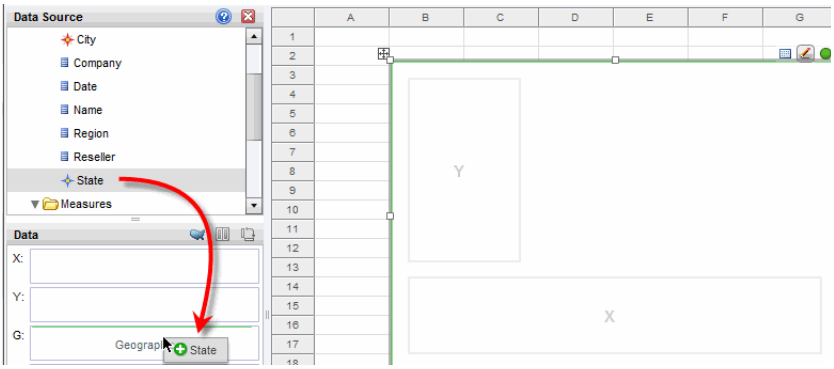
e. To resolve any unmatched names, press the red question mark. This opens the 'Geographical Mapping' dialog box, which allows you to manually match place names. Make any required matches and press the 'Apply' button.

f. To modify the settings for an existing geographical field, right click the field, and select 'Edit Geographic' from the context menu. This opens the geographical level editor described above.



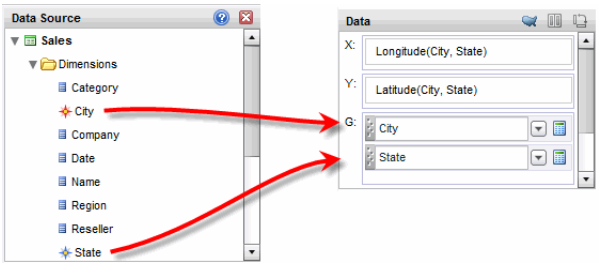
To remove the geographical status of a field, right click the field, and select 'Clear Geographic' from the context menu.

4. From the **Data Source** panel in the Chart Editor, drag a geographic dimension onto the 'Geographic' field in the **Data** panel.

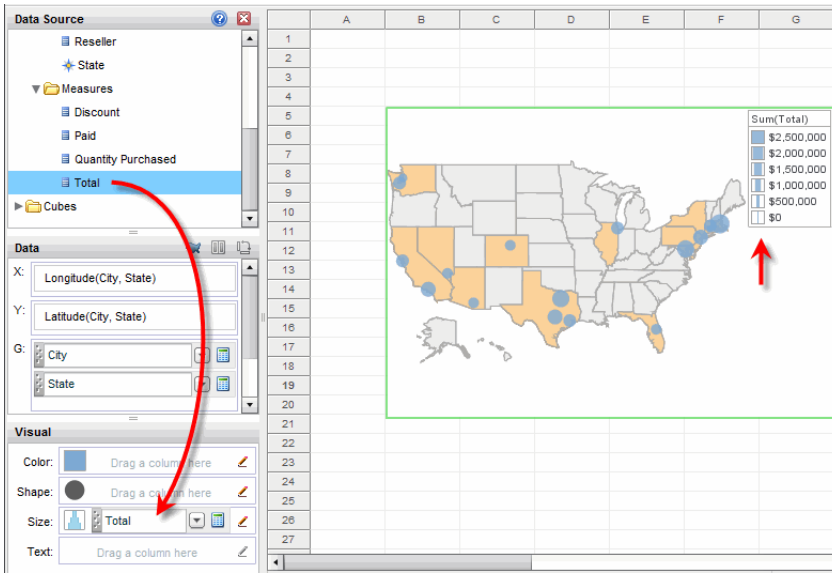


The chart updates to display the locations contained in the geographic field.

5. Repeat the previous step to bind additional 'Geographic' levels. For example, you can bind 'City' and 'State' fields.

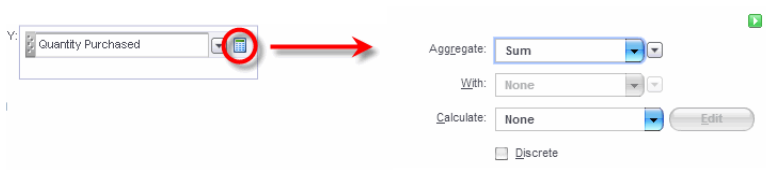


6. From the **Data Source** panel, drag a measure onto one of the fields in the **Visual** panel ('Color', 'Shape', 'Size', or 'Text').

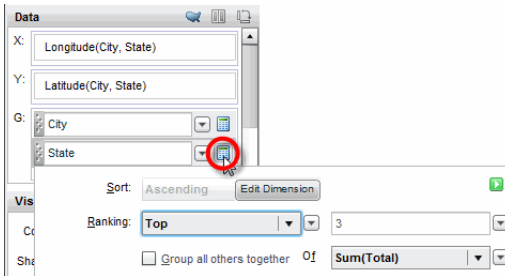


This displays the measure on the chart using the visual coding you selected (color, shape, or size), and automatically creates a corresponding legend.

7. Press the 'Edit Measure' button, and select the desired aggregation method for the measure.



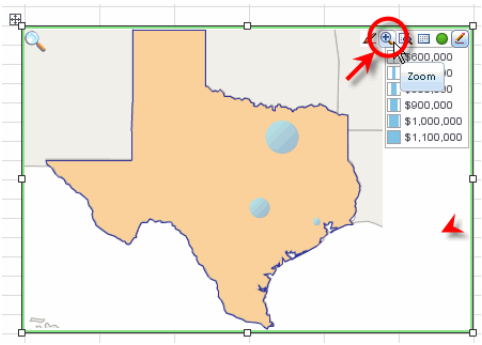
8. (Optional) For any geographical fields that you specified, press the 'Edit Dimension' button, and select the desired ranking for the geographical data. For example, you can display just the top three states according to the measure.



9. (Optional) To create a particular trajectory through the points on the chart, drag a *path* field to the 'P' region. The values of this field

are used to assign the sort order for the plotted data so that connecting lines are drawn between points that are adjacent in the *path* sort order.

10. (Optional) Set the initial map view by selecting a region, and pressing the 'Zoom' button.



For complete information about creating maps, see *Map Charts* in *Advanced Dashboard Design*.

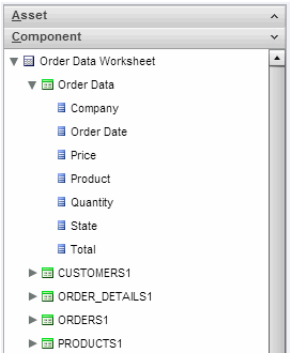
Adding a Selection (Filter)

To allow a user to filter the data displayed in the dashboard, add Selection components. A Selection component automatically filters the data in all tables, charts, and output components that are based on the same data block.

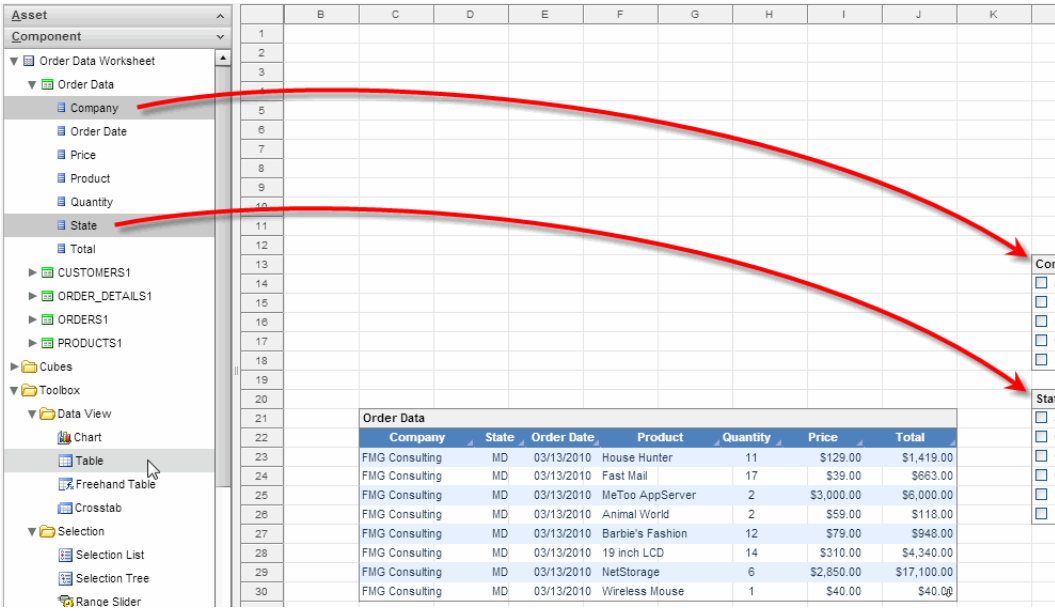
To add a Selection component, follow the steps below:

1. If necessary, create a new Viewsheet. See *Creating a New Dashboard*.

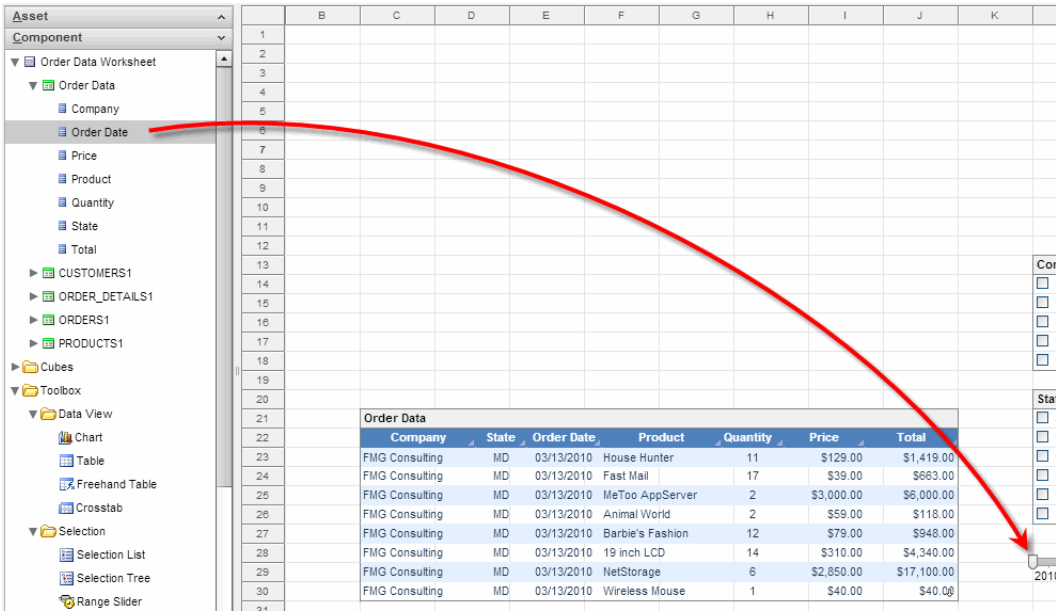
2. In the **Component** panel, expand the data block that contains the field you want to filter.



3. Drag the field you want to filter to an empty region in the Viewsheet grid. For a string-type field, this creates a Selection List.

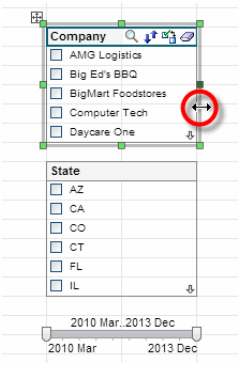


For a date-type or numerical-type field, this creates a range slider.

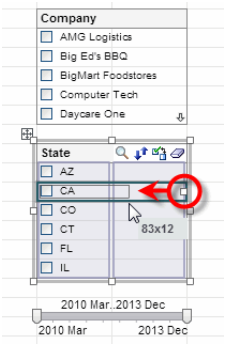


To filter dates more precisely, use a Calendar component. See *Calendar* in *Advanced Dashboard Design* for details.

4. Resize the Selection elements as desired.



5. To display multiple columns in a Selection List, click any data cell to select it. Then drag the handle on the right side of the cell to change the cell size. (Hint: You may need to wait a second for the scroll bar to fade before you can grab the handle.)



See *Selection Components* in *Advanced Dashboard Design* for full information about Selection components. See *Filtering with a Selection List* for information on how to use Selection Components to filter the dashboard.

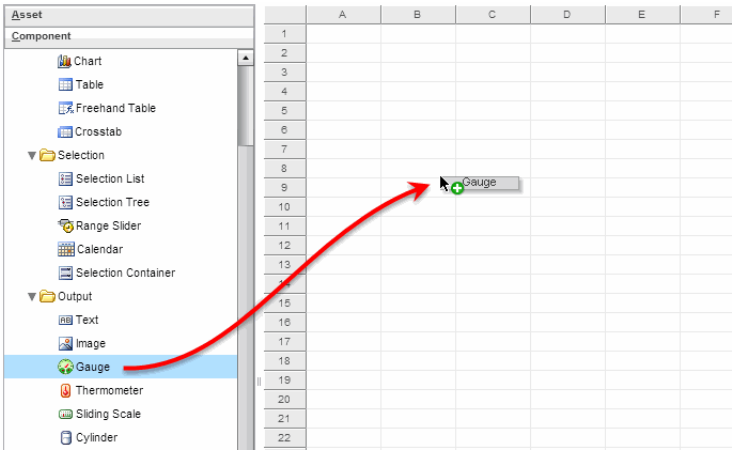
Adding an Output

An Output component allows you to display a single aggregated value (for example, a grand total) in a visually prominent manner. To add an Output component, follow the steps below:

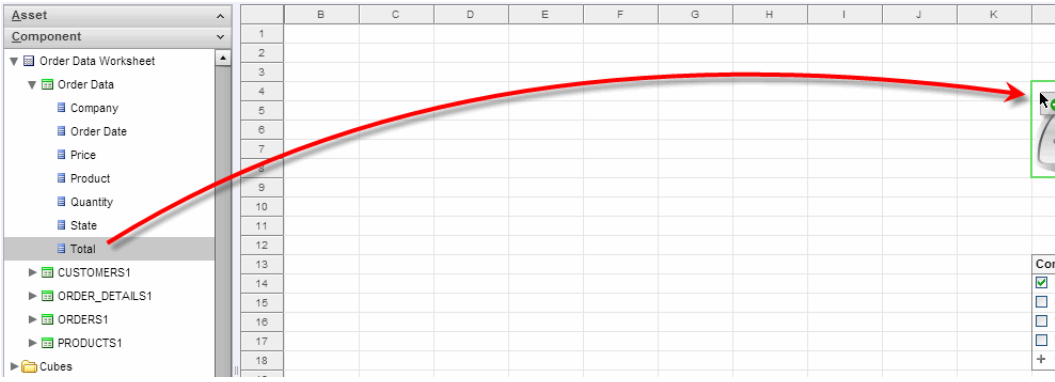
```
Note: The Gauge, Cylinder, Thermometer, and Slider are functionally
identical. They differ in appearance only.
```

1. If necessary, create a new Viewsheet. See *Creating a New Dashboard*.

2. From the **Component** panel, drag the desired Output component onto the Viewsheet grid.



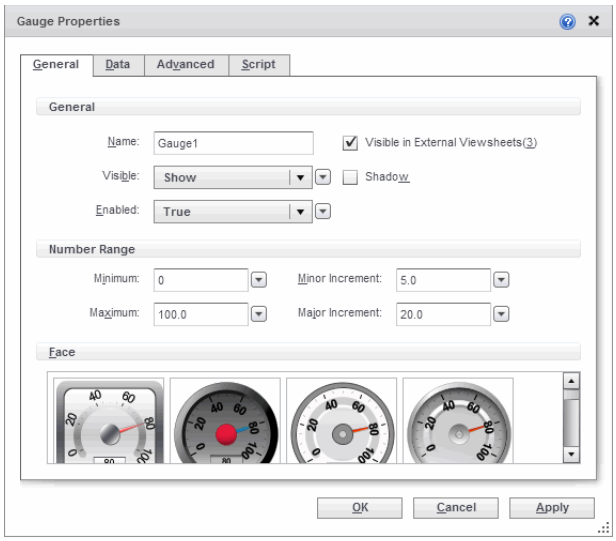
3. From the **Component** panel, drag a data block measure onto the Output component. (A green outline informs you when you can drop the field.)



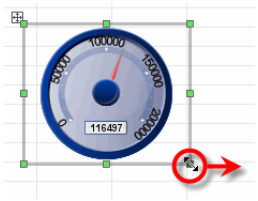
This binds the measure to the Output component so that the component displays the aggregated value of the measure.

4. Right-click on the Output component, and select ‘Properties’ from the context menu. This opens the ‘Properties’ dialog box.

5. Select the **General** tab of the ‘Properties’ dialog box, and choose a desired face. Press ‘OK’.



6. Enlarge the component as desired by dragging on any of the component borders.



See *Output Components* in *Advanced Dashboard Design* for full information about outputs.

Adding an Input

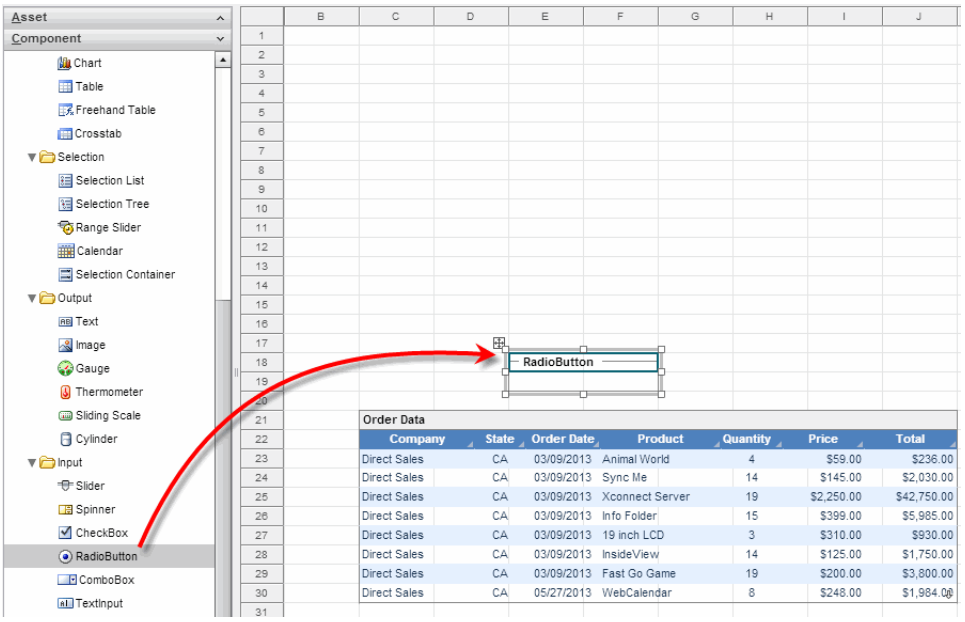
An Input component allows the user to dynamically control properties of other components. In this example, you will use an Input component to allow the user to change the measure displayed on a chart.

An Input component can also pass a value to the underlying Data Worksheet. (See *Input Components* in *Advanced Dashboard Design* for more information.)

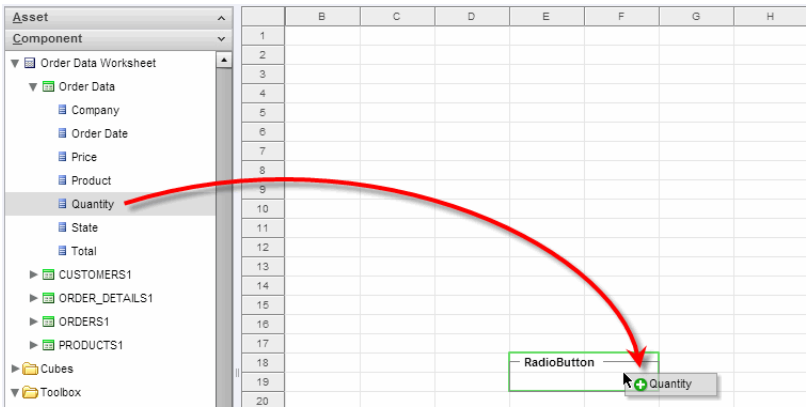
To add an Input component, follow the steps below:

1. Create a new Viewsheet, if needed. See *Creating a New Dashboard*.

2. From the **Component** panel, drag a RadioButton component into the Viewsheet grid.



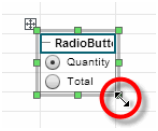
3. From the **Component** panel, drag a data block measure onto the RadioButton. (Look for the green highlight that indicates the drop zone.)



This adds the field name as an option in the RadioButton.

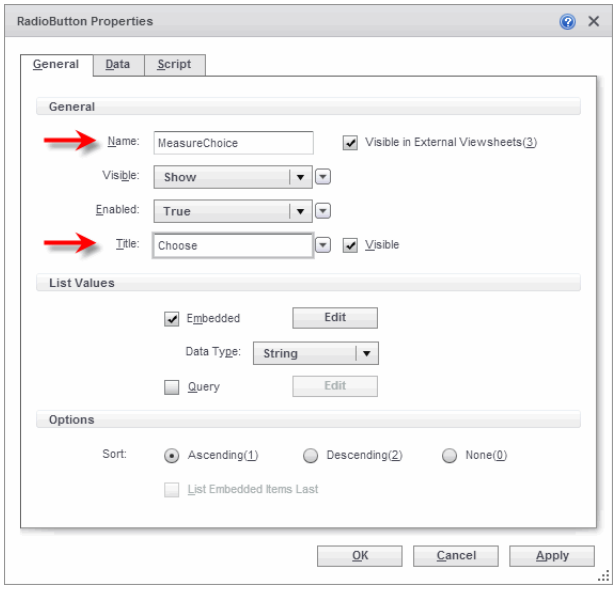
4. Repeat the above step to add other field names as additional options.

5. Resize the RadioButton as desired.

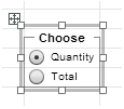


6. Right-click the RadioButton and select ‘Properties’ from the context menu. This open the ‘RadioButton Properties’ dialog box.

7. Set the ‘Name’ property to describe the role of the Input (e.g., “MeasureChoice”). Set the ‘Title’ property to “Choose”. Press ‘OK’.



The ‘Name’ is a unique component ID. (You will need to refer to the RadioButton by its name in a later step.) The ‘Title’ is simply the text that appears in the title bar of the component.

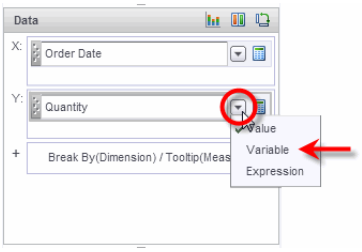


To configure a Chart so that the RadioButton controls the measure displayed on the Chart Y-axis, continue with the steps below. See *Adding a Chart* for information about how to create a chart.

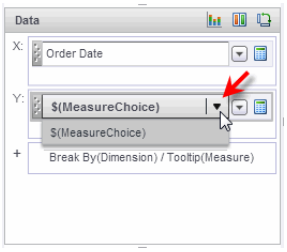
8. Open the Chart Editor by pressing the ‘Edit’ button at the top-right corner of the chart.

```
Many dashboard properties can be controlled by Input components in
the same manner.
```

9. In the Chart Editor, expand the pop-up menu next to the field on the ‘Y’ axis, and select ‘Variable’. The ‘Variable’ option allows the property to be set by an Input component.



10. Select the name of the desired Input Control from the menu.



By selecting the name of an Input control, you are specifying that the value the user chooses in the Input will determine the field that is displayed on the chart Y-axis.

11. Use the Input control to toggle between the different field names and observe how the chart Y-axis adapts to display the corresponding measure.

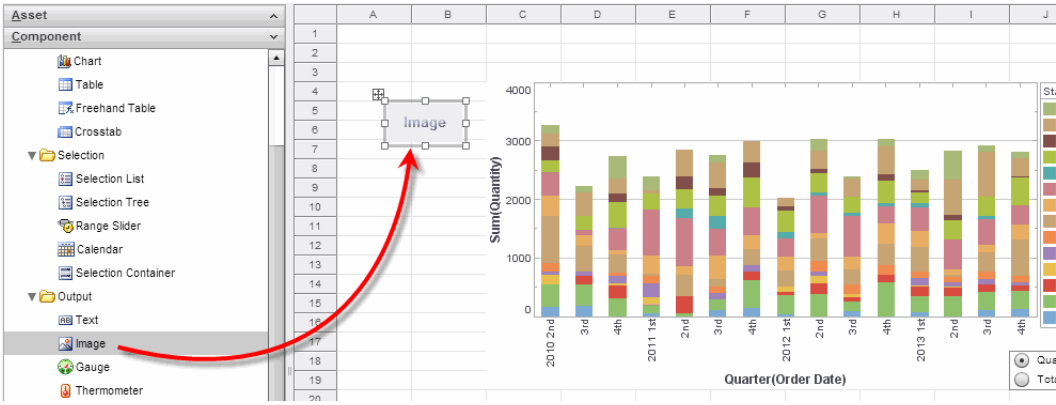
See *Input Components* in *Advanced Dashboard Design* for full information about inputs.

Adding Decoration

To add text, images, and background shapes to the dashboard, follow the steps below:

1. If necessary, create a new Viewsheet. See *Creating a New Dashboard*.

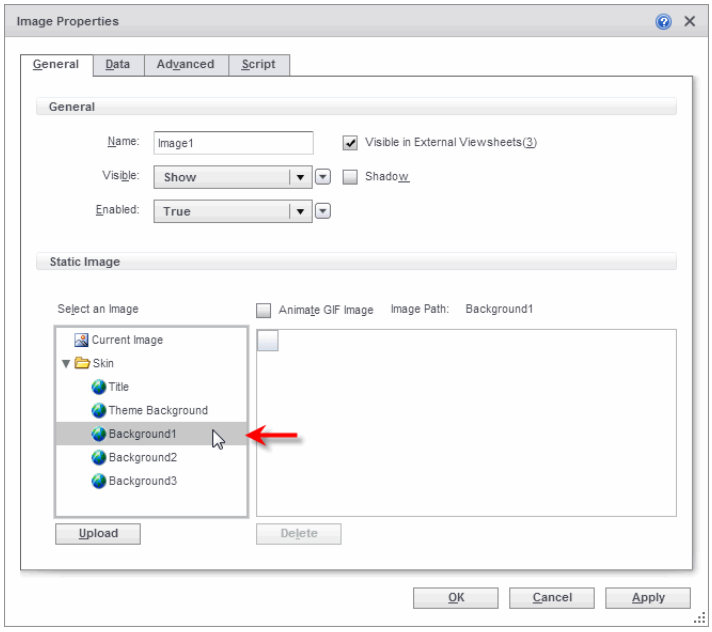
2. From the **Component** panel, drag an Image component into an empty area of the Viewsheet grid. In this example, the image will be used as a background.



3. Right-click on the Image component, and select ‘Properties’ from the context menu. This opens the ‘Image Properties’ dialog box.

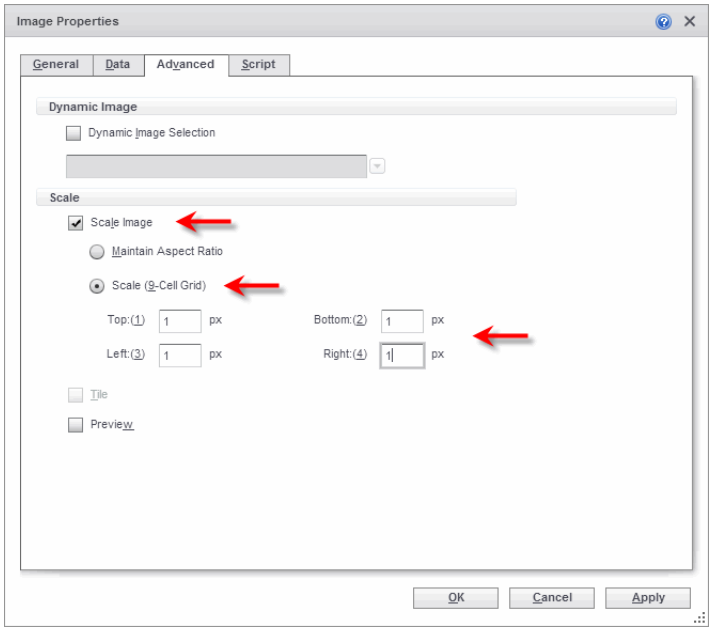
To upload a local image instead, press the ‘Upload’ button.

4. Under the **General** tab, expand the ‘Skin’ folder and select the desired skin.



5. Select the **Advanced** tab.

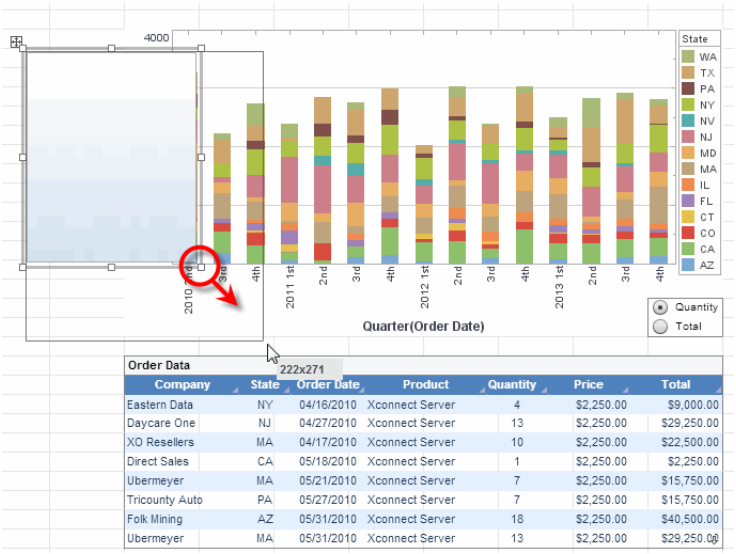
6. Check the 'Scale Image' box, and select the 'Scale (9-Cell Grid)' option. Enter "1" in each of the 'Top', 'Right', 'Bottom', and 'Left' fields.



This allows the interior of the image (i.e., the gradient fill) to scale without scaling the 1-pixel border around the image. (The 1-pixel gray border would look strange if scaled with the rest of the image.)

7. Press the ‘OK’ button to dismiss the dialog box.

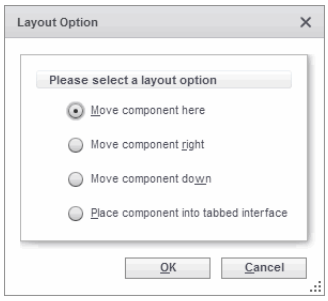
8. Drag the borders of the Image component to resize the image so that it is large enough to provide a background for the dashboard.



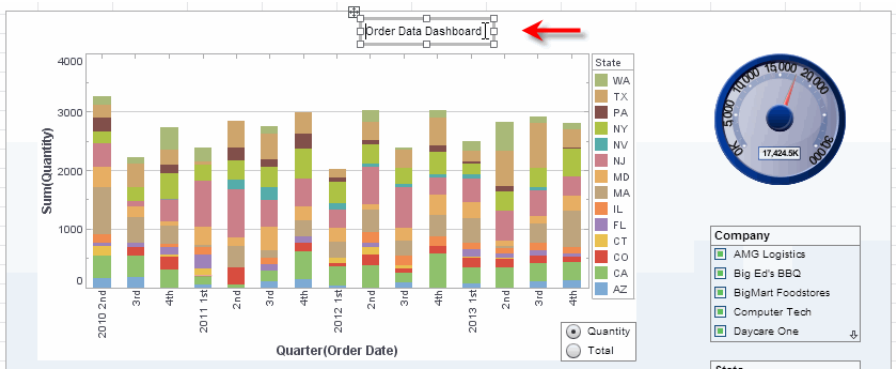
9. Right-click on the Image component and select ‘Send to Back’ to move the Image component behind all other components. Continue resizing the Image component as needed to create the desired background.



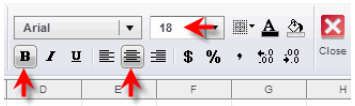
10. From the **Component** panel, drag a Text component into the Viewsheet. (If you see the following 'Layout Option' dialog box, select 'Move component here' and press 'OK'.)



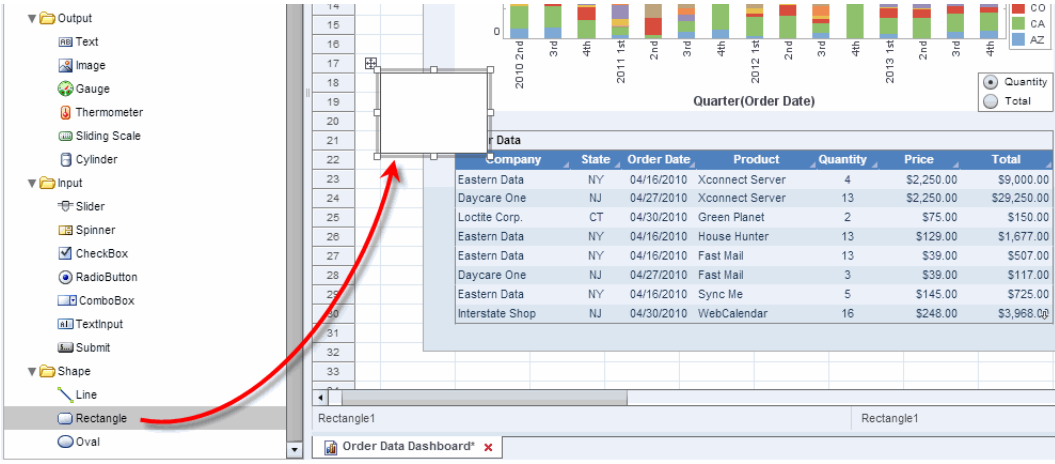
11. Double-click on the Text component to edit the text.



12. Use the toolbar buttons to adjust the appearance of the text.

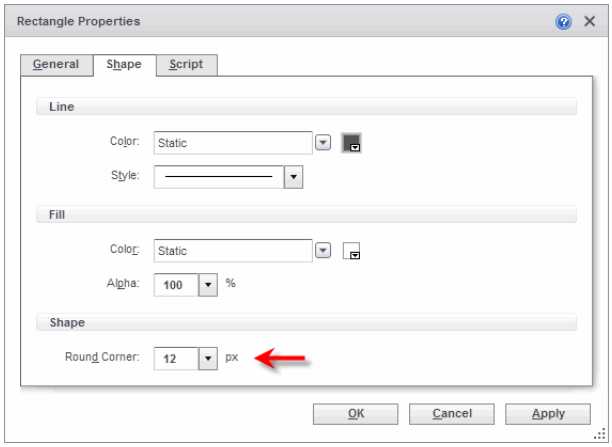


13. From the **Component** panel, drag a Rectangle component to an empty area of the Viewsheet grid.

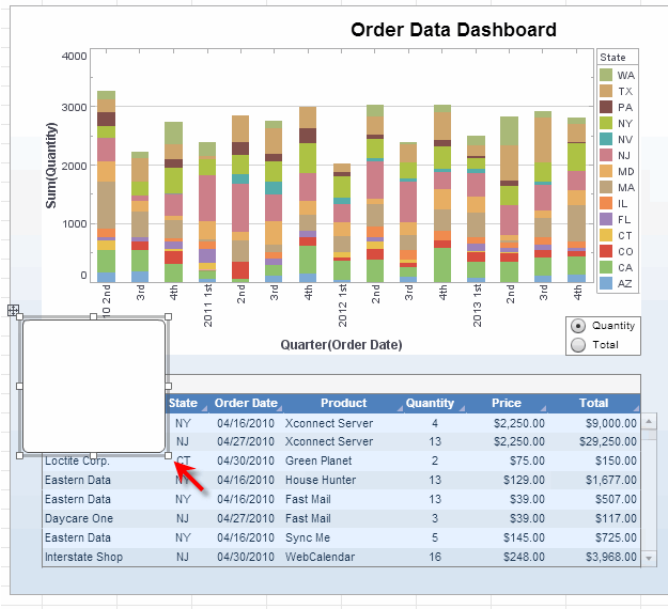


14. Right-click the Rectangle component and select ‘Properties’ from the context menu. This opens the ‘Rectangle Properties’ dialog box.

15. Select the **Shape** tab. Set the ‘Round Corner’ property to soften the corners of the rectangle. (Larger values indicate softer corners.)



16. Drag the Rectangle to an appropriate position to provide a background for a group of components.



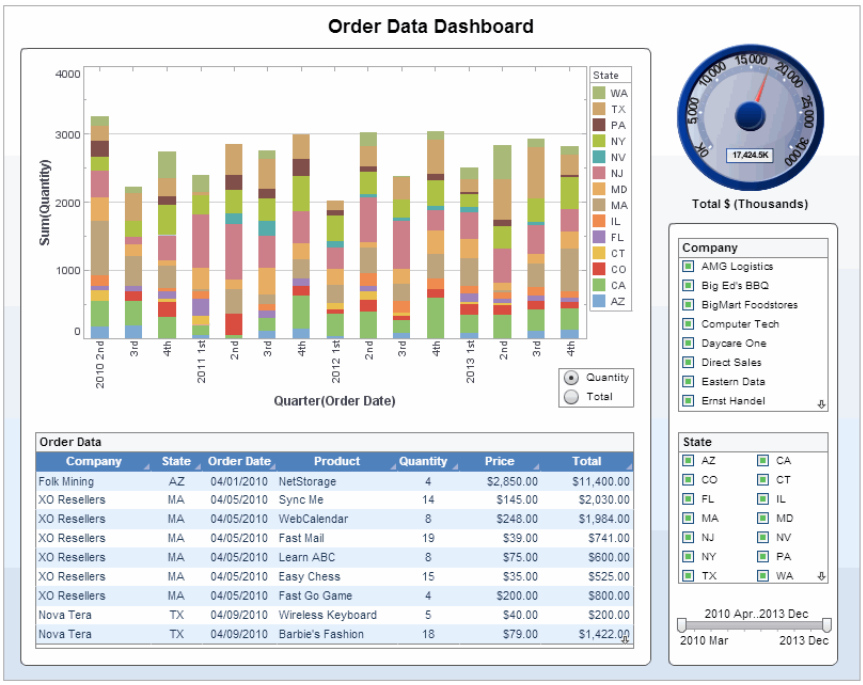
17. Right-click the Rectangle and select ‘Send to Back’ from the context menu. If the Rectangle disappears behind other components, then repeat the same procedure to send those components to the back.

See *Shape Components*, *Text*, and *Image* in *Advanced Dashboard Design* for full information about these components.

Previewing and Saving a Dashboard



Although a dashboard is generally fully functional in design view of Visual Composer, sometimes it is useful to preview the dashboard exactly as users will see it. To preview the dashboard, press the ‘Preview’ button on the toolbar.

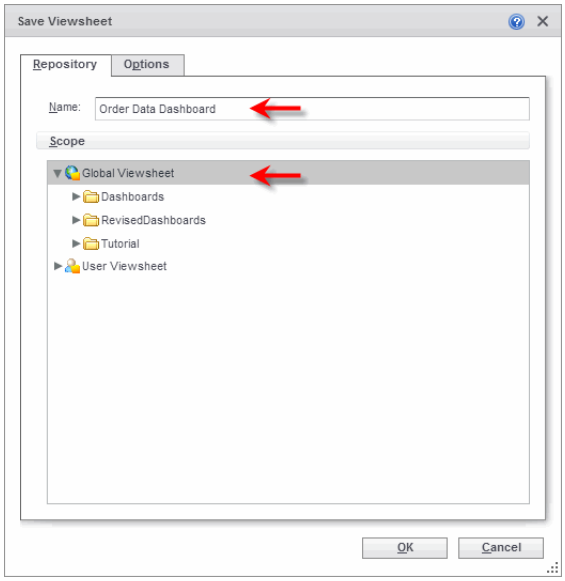


To save a dashboard, follow the steps below:

1. Press the ‘Save’ button in the toolbar. This opens the ‘Save Viewsheet’ dialog box.

2. In the ‘Name’ field, enter the name under which the dashboard should be saved.

3. To save into a public location, select the ‘Global Viewsheet’ node or a subfolder. In this case, the dashboard will be available to all users who have read permission for the selected location.



To save into a private location, select the ‘User Viewsheet’ node and press ‘OK’. The dashboard will only be accessible to its creator.

Viewsheets are stored in a file called *asset.dat* in the server repository.

Boosting Dashboard Performance

```
Only users who have administrator privileges can change server
settings.
```

The best method for boosting dashboard performance is to create a materialized view. A *materialized view* is a caching strategy that pre-aggregates and stores the data required by a dashboard (Viewsheet) so that the dashboard can respond more quickly at runtime. When a materialized view exists for a Viewsheet, the server attempts to query the cached materialized view rather than querying the database. This can significantly reduce the run time for certain queries, which increases dashboard responsiveness.

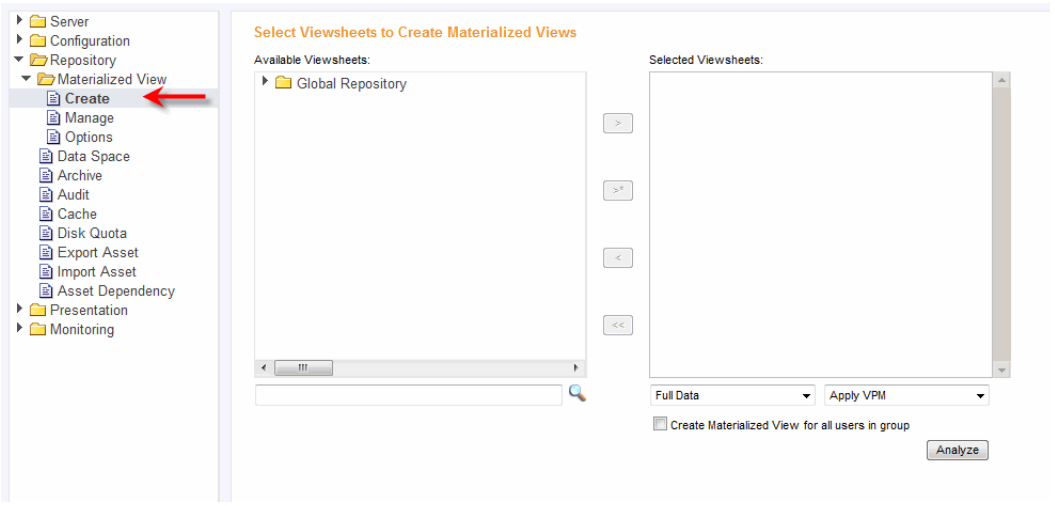
The following sections provide a quick description of how to create and manage materialized views. For complete information, see *Materialized Views* in *Advanced Server Administration*.

Creating a Materialized View

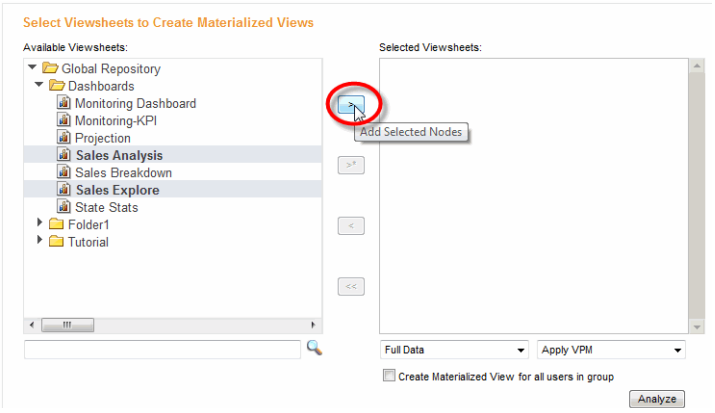
`Only users who have administrator privileges can log into Enterprise Manager.`

To create the materialized view for a Viewsheet, follow the steps below:

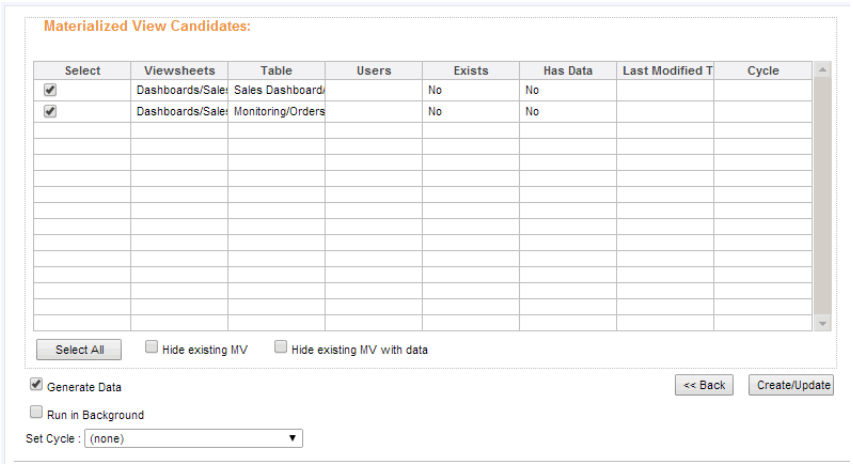
1. Log into Enterprise Manager.
2. Expand the 'Repository' > 'Materialized View' node under the **Server** tab, and select 'Create'.



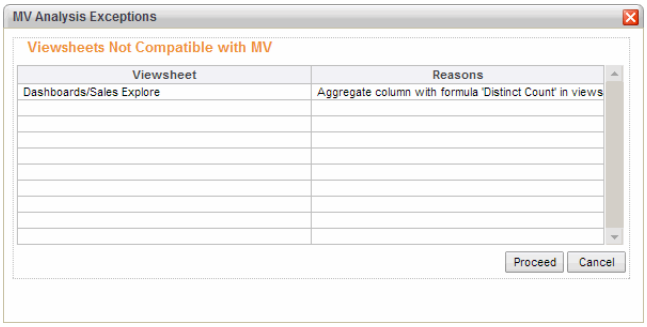
3. Select one or more Viewsheets or folders from the 'Available Viewsheets' panel, and press 'Add Selected Nodes'. This adds the selected items to the 'Selected Viewsheets' panel.



4. Press the 'Analyze' button. This opens the 'Materialized View Candidates' page containing a list of Viewsheets that can be materialized.

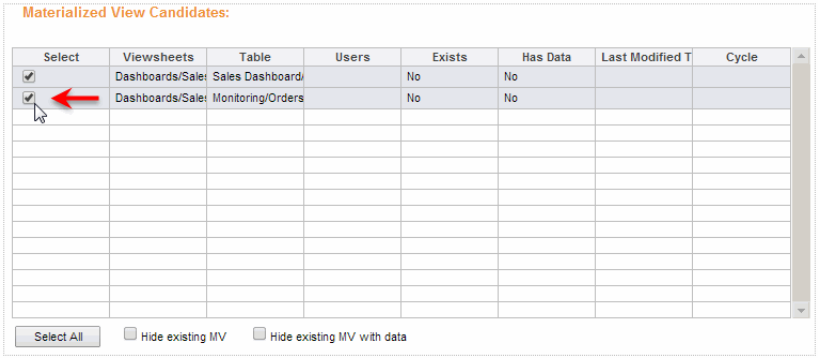


If there are design aspects of a Viewsheet or Data Worksheet that prevent the generation of a materialized view, the 'MV Analysis Exceptions' dialog box is shown.



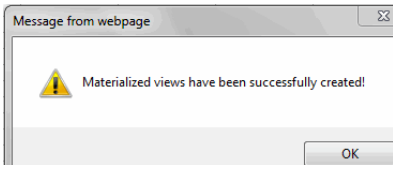
Press 'Proceed' to continue with the materialization process for any materializable Viewsheets, or press 'Cancel' to abort the materialization process.

5. Check the boxes in the 'Select' column to select the Viewsheets for which you want to create a materialized view. To select all Viewsheets in the table, press the 'Select All' button below the table.



6. In the 'Select Cycle' column of the table, choose the cycle on which the materialized view should be regenerated.

7. Press the 'Create/Update' button. If you select the 'Generate Data' option, this will immediately start the generation process and notify you upon completion.



   If you select the 'Run in Background' option, the materialized view will be scheduled to begin generation. You can check the status of the task under the **Schedule** tab.

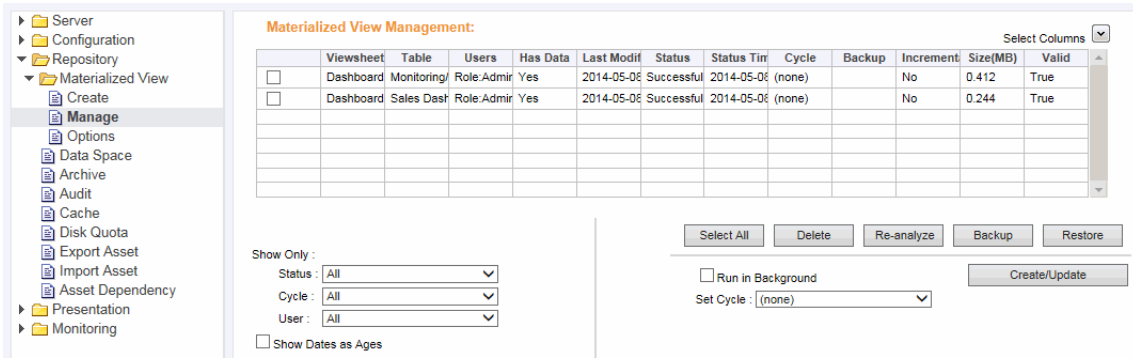
To manage existing materialized views, see *Managing Materialized Views*.

See Also

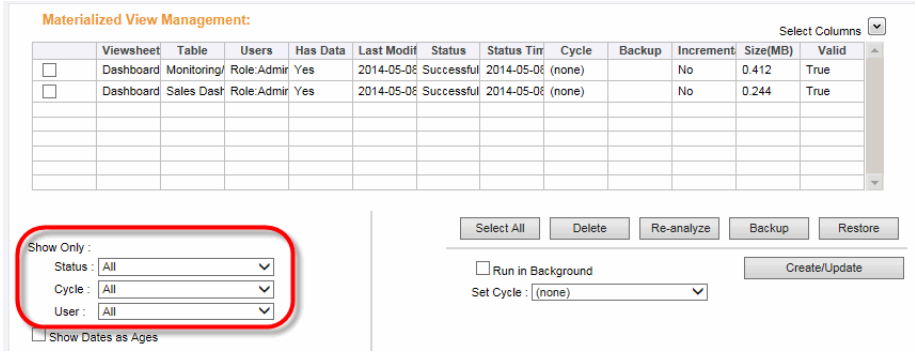
See *Materialized Views* in *Advanced Server Administration* for complete information on creating materialized views.

Managing Materialized Views

To manage existing materialized views, select the 'Repository' > 'Materialized View' > 'Manage' page under the **Server** tab.



The table lists all existing materialized views. To restrict the items shown in the table by status, cycle, or user, make the desired selections from the left-side menus:



To delete a materialized view (for example, if changes have been made to the underlying database schemas), select the materialized view in the table and press 'Delete'.

For complete information on managing materialize views, see *Managing Materialized Views* in *Advanced Server Administration*.

See Also

Managing Materialized Views, in *Advanced Server Administration*, for more information about creating and editing Viewsheets.

Interacting and Collaborating

The following sections explain the basics of working with dashboards.

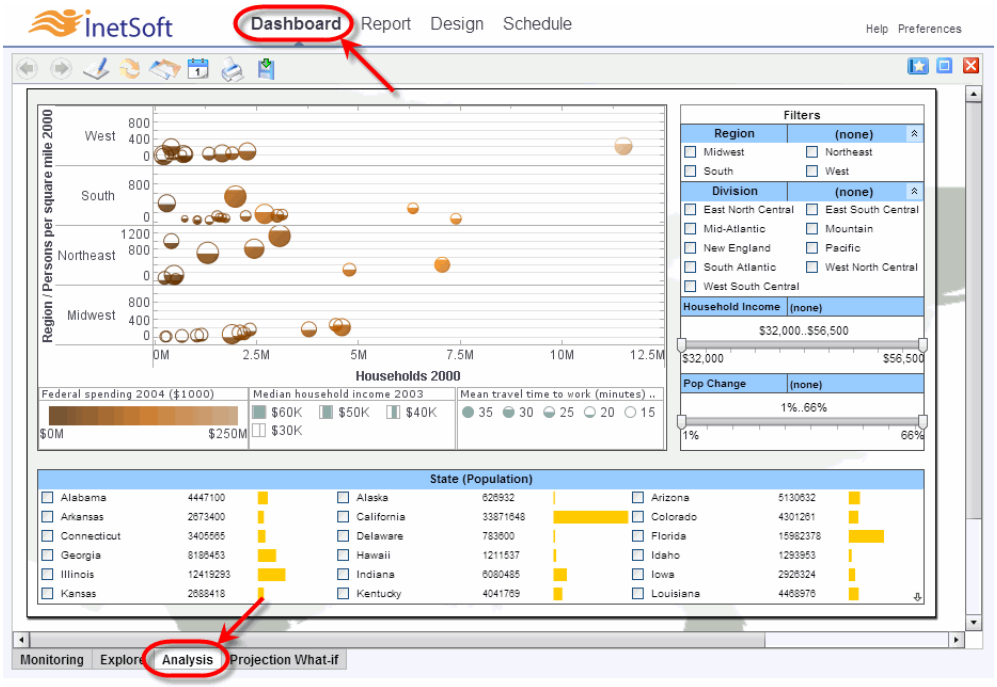
Interacting with Dashboards

The following sections discuss basic aspects of interacting with dashboards, including filtering with Selection List and Calendar components.

Opening a Dashboard

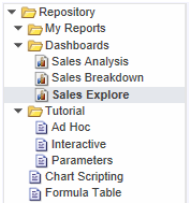
To open a dashboard from the **Dashboard** tab, follow the steps below:

1. Open the User Portal. (See *Using the Portal* for more information.)
2. Select the **Dashboard** tab at the top of the User Portal.
3. Select the tab for the dashboard that you want to view. The tabs may be located either at the top or bottom of the screen.



To open a dashboard from the **Report** or **Viewer** tab, follow these steps:

1. Open the User Portal. (See *Using the Portal* for more information.)
2. Select the **Report** or **Viewer** tab at the top of the User Portal.
3. In the tree on the left side of the Portal, expand the folder containing the desired dashboard.

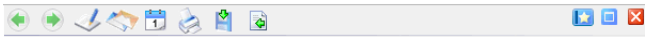


4. Click on the dashboard that you want to view (for example, 'Sales Explore' in the 'Dashboards' folder).



Using the Dashboard Toolbar

The dashboard toolbar has the following buttons:





Previous: Undoes the most recent action.



Next: Restores the most recent action that was undone.



Edit: Opens the current dashboard in Visual Composer. For more information about creating and modifying dashboards using Visual Composer, see *Advanced Dashboard Design*.



Refresh: Updates the dashboard with current data. (Not available for dashboards that use materialized views.)



Email: Sends an export of the dashboard to another person. See *Emailing a Dashboard* for more information.

Schedule: Configures a dashboard for automatic generation and delivery. See *Scheduling a Dashboard* for more information.

Print: Prints the dashboard on any available printer.

Export: Exports a copy of the dashboard to Excel, PowerPoint, or PDF format. See *Exporting a Static Dashboard* for detailed instructions.

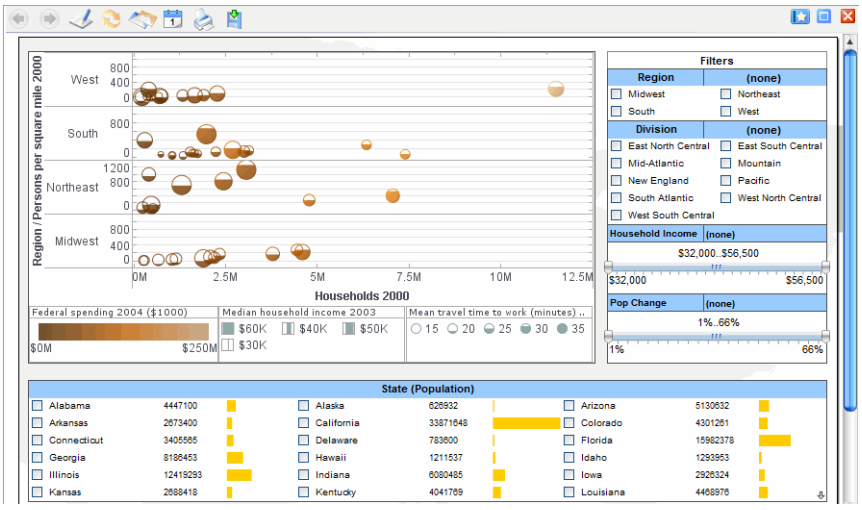
Bookmark: Opens the 'Bookmark' menu, which allows you to save a bookmark or load a previously-saved bookmark. See *Bookmarking a Dashboard* for more details.

Annotation: Toggles annotations on and off. See *Annotating a Dashboard* for more details.

Filtering with a Selection List

Selection components allow you to filter the data that is displayed in the dashboard and create custom views of the dataset. The following example illustrates how to use the Selection Lists in the sample 'Analysis' dashboard.

1. Log into the User Portal. (See *Logging into the Portal*.)
2. Press the **Dashboard** tab at the top of the User Portal.
3. Select the **Analysis** tab at the bottom of the Portal to open the sample 'Analysis' dashboard.

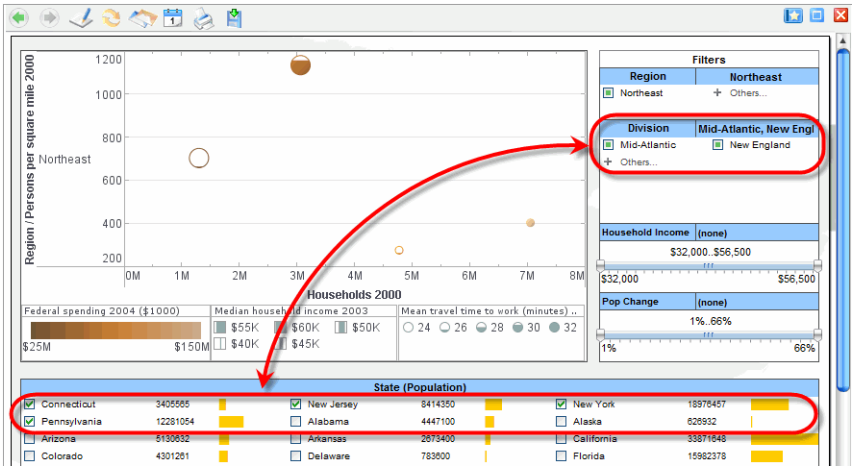


4. Press the 'Bookmark' button at the top-right of the dashboard, and select the '(Home)' option. This ensures that you are starting from the dashboard's original configuration.

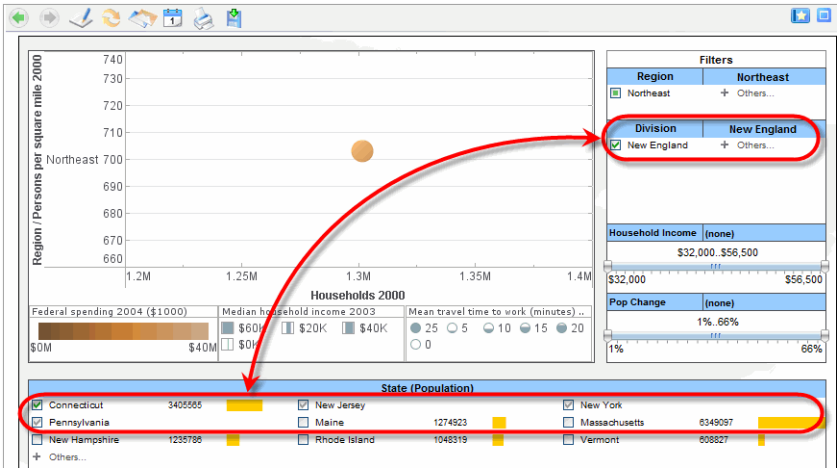
Note: Some Selection Lists submit automatically when you make a selection. To make multiple selections on this kind of Selection List, Ctrl-click when selecting options.

5. Select a few states (e.g., 'New Jersey', 'New York', 'Pennsylvania', 'Connecticut') in the 'State' Selection List.

   Observe how the Selection List filters the data in the chart, and note that the 'Division' selection list now shows two items with green squares, 'Mid-Atlantic' and 'New England'. The remaining divisions are listed under 'Others'. The two green divisions are those that are *compatible* with the current 'State' selection. (The selected states are in the Mid-Atlantic and New England divisions.)



6. Filter the results further based on division. Select 'New England' in the 'Division' list. When you make this selection in the 'Division' selection list, the 'State' selection list automatically updates.



   Note the following features of the 'State' selection list:

   a. Connecticut is shown with a green check mark. This indicates that you explicitly selected 'Connecticut' and the records for Connecticut were returned in the result set.

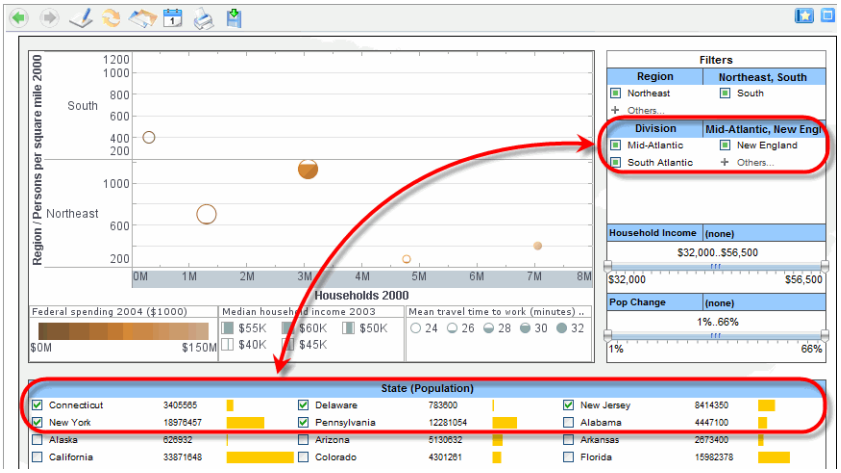
b. Three states are shown with gray check marks (New Jersey, New York, Pennsylvania). This indicates that you explicitly selected these states but their records are *not* included in the result set because of an incompatible choice in another selection list. In this case, you explicitly selected the 'New England' division, so only records for New England states (i.e., Connecticut) are actually included.

c. Five states are shown with empty boxes (Maine, Massachusetts, New Hampshire, Rhode Island, Vermont). This indicates that records for these states are *not* included, but these choices are compatible with your other selections. You can select these choices to widen your result set.

d. All other states are listed under 'Others'. This indicates that these records are *not* included and are *not* compatible with your other selections.

7. Expand the 'Others' node in the 'State' selection list. Notice that all incompatible states have a gray 'X' icon.

8. Select 'Delaware'.



This clears the 'New England' selection from the 'Division' list. Whenever you make a selection from an 'Others' list, this resets any incompatible selections in other lists. In this case, the 'Division' list is reset to show only divisions compatible with the five currently selected states.

For more information about the different icons in Selection Lists, see *Selection Lists*.

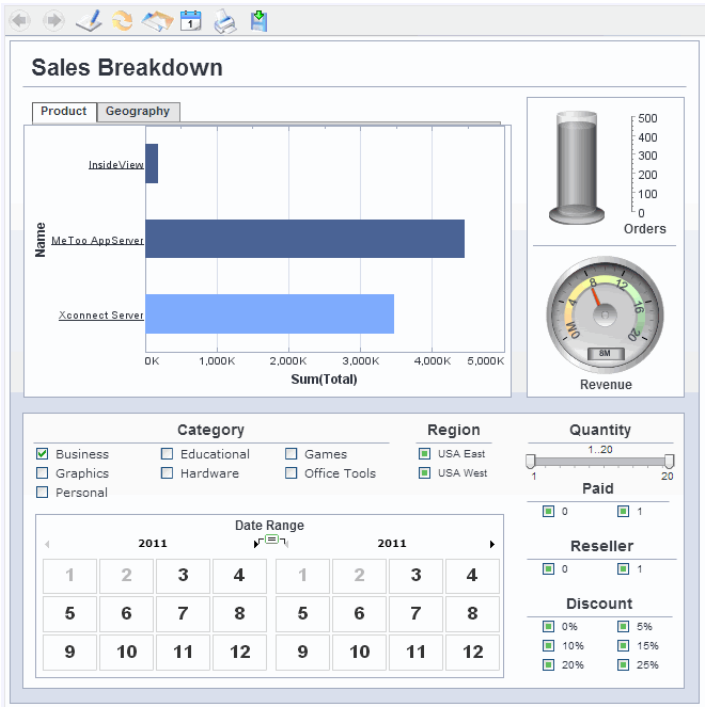
Filtering with a Calendar

The Calendar component provides a calendar interface that allows you to filter data based on a range of dates or a comparison of dates. The following example illustrates how to use the Calendar in the sample 'Sales Breakdown' dashboard.

1. Log into the User Portal. (See *Logging into the Portal*.)

2. Press the **Report** or **Viewer** tab at the top of the User Portal.

3. Expand the 'Dashboards' folder in the Repository listing, and select the 'Sales Breakdown' dashboard.



4. Press the 'Bookmark' button at the top-right of the dashboard, and select the '(Home)' option. This ensures that you are starting from the dashboard's original configuration.

5. First, use the Calendar to compare sales for Feb. 2013 against sales for Feb. 2014. To do this, follow the steps below

   a. Press the arrow buttons next to the year labels to advance the left-side year to 2013 and the right-side year to 2014.



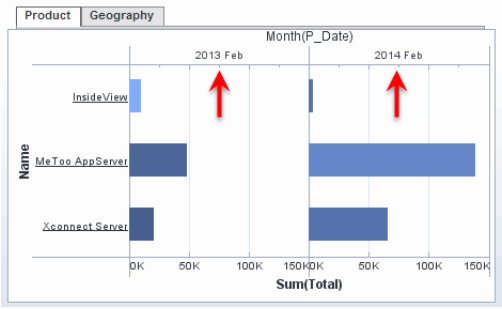
b. Select Feb. 2013 and Feb. 2014 by clicking “2” on both sides of the calendar.



c. Press the green ‘Apply’ button to apply the month comparison.



Observe that the chart updates to show a comparison between the selected months.

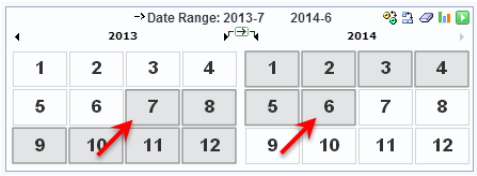


6. Next, use the Calendar to restrict the data to sales between July 2013 and June 2014. To do this, follow the steps below:

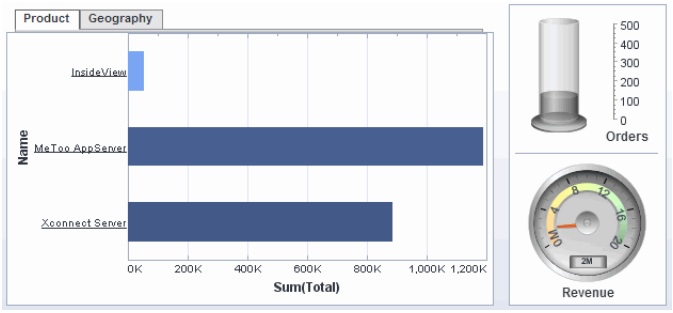
a. Press the ‘Switch to Date Range Mode’ button in the Calendar mini-toolbar.



b. Click to select month “7” in year 2013 and month “6” in year 2014.

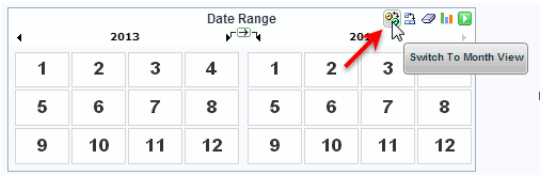


c. Press the green ‘Apply’ button to filter the dashboard to display only data from the selected date range.

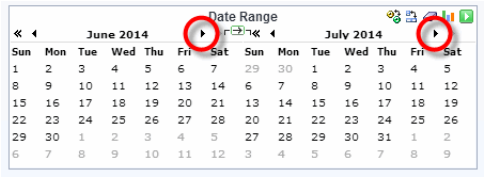


7. Finally, use the Calendar to restrict the data to sales between the third week in June 2014 and the second week in July 2014. To do this, follow the steps below:

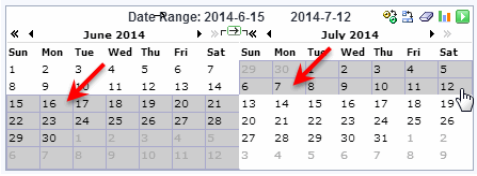
a. Press the ‘Switch to Month View’ button in the Calendar mini-toolbar.



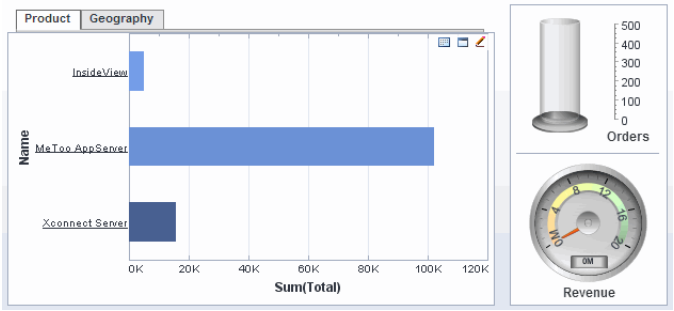
b. Press the arrow buttons until June 2014 is displayed on the left side and July 2014 is displayed on the right side.



c. Click to select the third week in June and click to select the second week in July.



d. Press the green ‘Apply’ button to filter the dashboard to display only data from the selected date range.

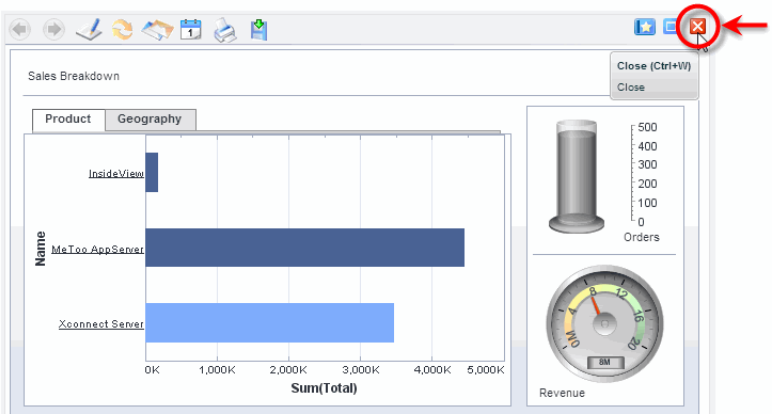


See Also

Calendars, for more details on end-user Calendar features.

Closing a Dashboard

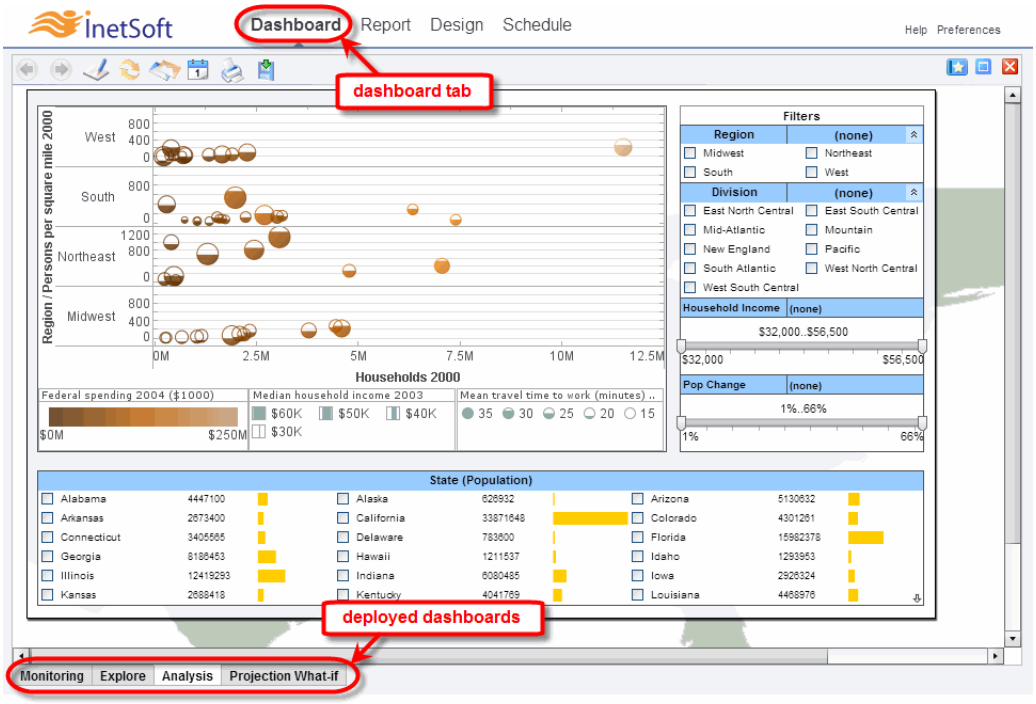
To close a dashboard, press the red ‘X’ in the toolbar, or simply open a different dashboard.



When you close a dashboard, you will be prompted to save your current settings (filter selections, menu choices, annotations, etc.) in a bookmark. If you choose to save your settings, you will be able to restore these settings from the bookmark at a later time. See *Bookmarking a Dashboard* for more details.

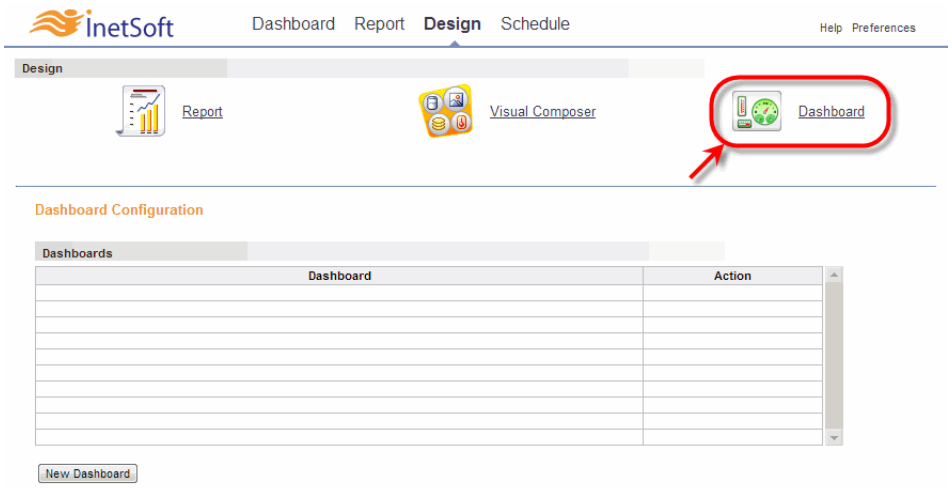
Deploying a Dashboard

You can arrange the dashboards displayed under the **Dashboard** tab of the User Portal, and add your own dashboards under this tab.



Each of the dashboards that appears under the **Dashboard** tab of the User Portal consists of one or more component dashboards called “Viewsheets”. To add a new dashboard to the **Dashboard** tab, follow these steps:

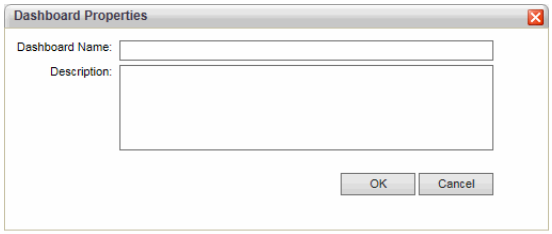
1. Select the **Design** tab in the User Portal, and press the ‘Dashboard’ button. This opens the ‘Dashboard Configuration’ page.



2. Press the ‘New Dashboard’ button under the ‘Dashboards’ table. This opens the ‘Dashboard Properties’ dialog box.

3. In the ‘Dashboard Name’ field, enter a name for the new dashboard. This name will be displayed on the dashboard’s tab in the Portal.

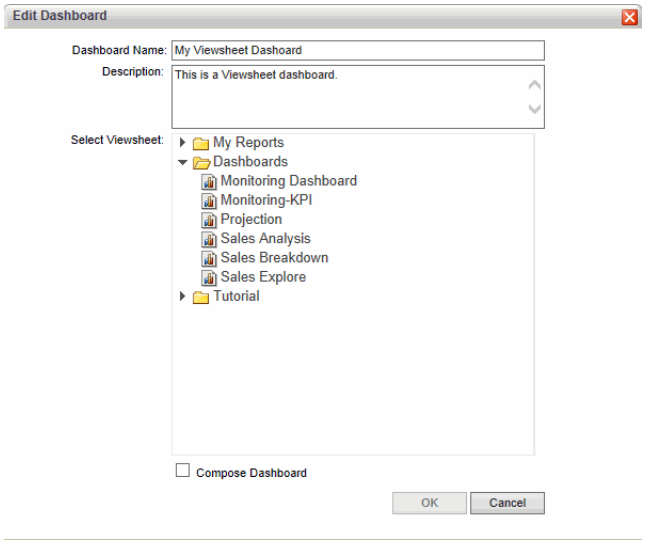
4. (Optional) Enter a description for the dashboard in the ‘Description’ field. This description is only visible when the dashboard is being edited.



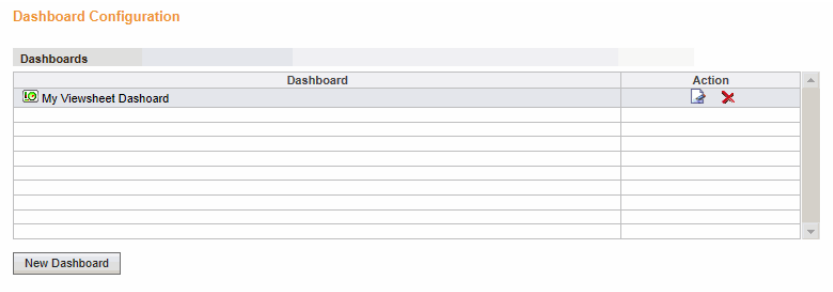
5. Press ‘OK’. This opens the ‘Edit Dashboard’ dialog box.

6. To specify a single existing Viewsheet to use as the dashboard, follow the steps below:

   a. Choose the desired Viewsheet from the ‘Select Viewsheet’ list.



b. Press ‘OK’. This closes the ‘Edit Dashboard’ dialog box, and returns you to the main ‘Dashboard Configuration’ page.



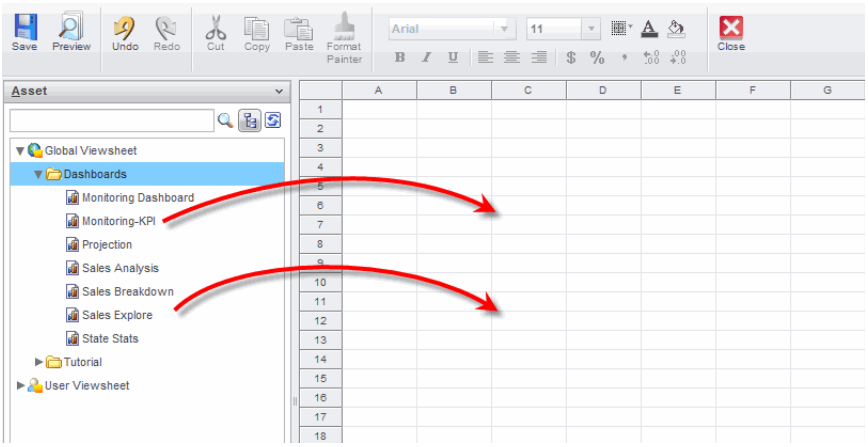
7. To compose a dashboard by combining several existing Viewsheets, follow the steps below:

   a. Select the ‘Compose Dashboard’ option at the bottom of the ‘Edit Dashboard’ dialog box.

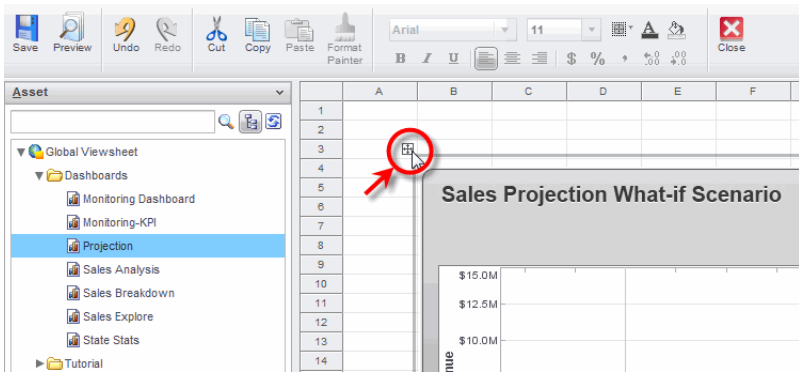
   b. Press ‘OK’. This opens the Visual Composer.

   c. From the **Asset** panel, drag the desired Viewsheets into the Viewsheet grid to compose the dashboard layout.

See *Nesting a Viewsheet* in *Advanced Dashboard Design* for more information about nesting one Viewsheet within another Viewsheet.

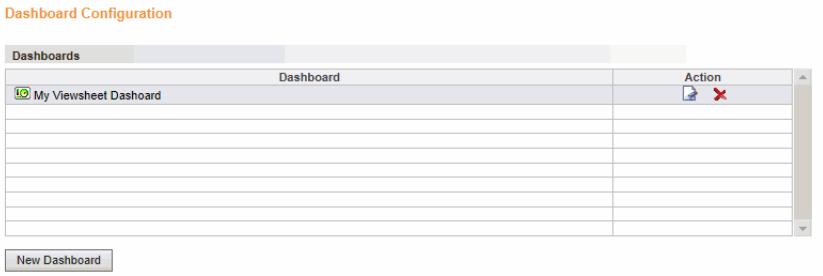


   d. Click and drag the “move” handle on a nested Viewsheet to position it on the grid.



e. Press the ‘Save’ button in the Visual Composer toolbar to save the composite dashboard.

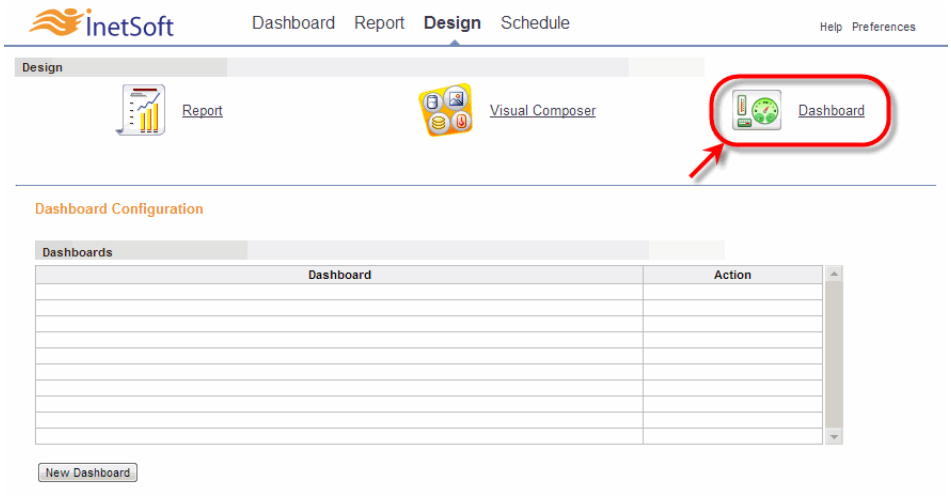
f. Press the ‘Close’ button in the Visual Composer toolbar to close the Visual Composer. This returns you to the main ‘Dashboard Configuration’ page.



The new dashboard is now listed in the ‘Dashboards’ table, and will be visible under the **Dashboards** tab of the Portal. See *Configuring Portal Preferences* to arrange the dashboards under the **Dashboards** tab.

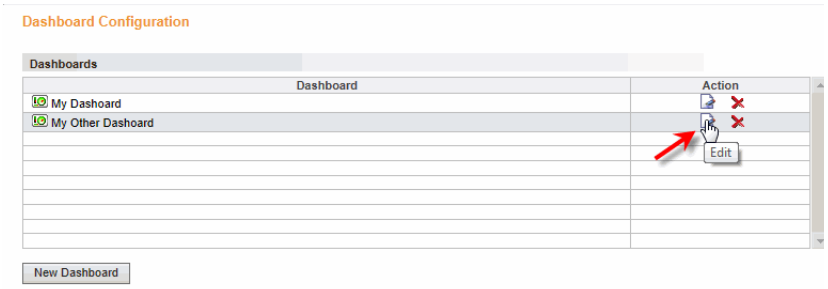
To edit a dashboard that you have added under the **Dashboard** tab, follow the steps below:

1. Select the **Design** tab in the User Portal, and press the ‘Dashboard’ button.



2. Locate the dashboard you wish to edit in the 'Dashboards' table. (This table lists all of the user-defined dashboards. It does not list global dashboards defined by the administrator.)

3. Press the 'Edit' button in the rightmost column for the dashboard you wish to edit.



This opens the 'Edit Dashboard' dialog box. Follow the instructions above to make the desired edits.

To delete a dashboard entirely, press the 'Delete' button in the rightmost column.

See Also

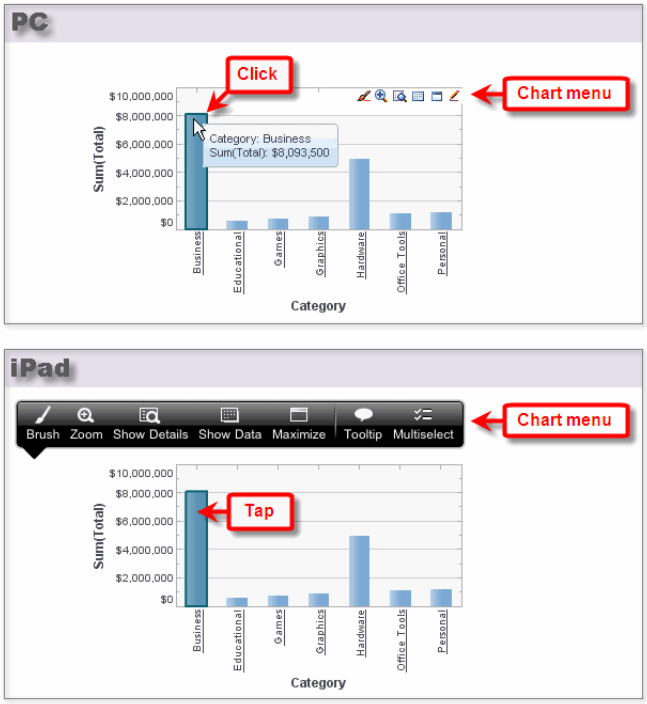
Creating a Dashboard, for information about creating a new dashboard.

Using a Dashboard on a Mobile Device

You can access dashboards on a mobile device (iPad, Android, etc.) in the same way that you access dashboards on a personal computer. To open the User Portal, simply point your browser to the URL that your administrator has provided. For example:

```
http://hostname:8080/sree/Reports
```

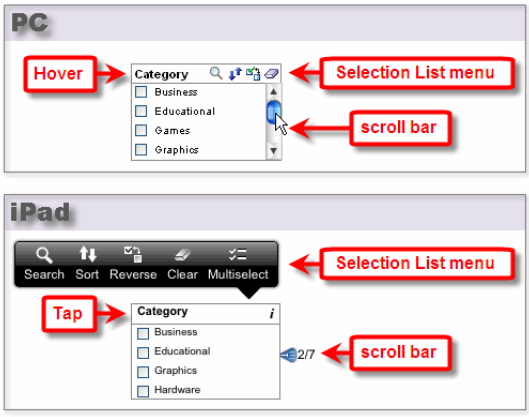
On iOS and Android devices, the dashboard interface is slightly modified to take advantage of these platforms. Component toolbars are displayed as a floating toolbars. Simply tap on the desired dashboard component to expand the component's toolbar. For example, the illustration below shows the Chart menu as it appears on a PC and an iPad.



To activate tooltips for a Chart, press the 'Tooltip' toolbar button. To activate multiple-selection (similar to Ctrl-click on a PC), press the 'Multiselect' toolbar button. To activate a hyperlink, *press and hold* the hyperlinked region.

To activate multiple-selection (similar to Ctrl-click on a PC), press the 'Multiselect' toolbar button.

The illustration below shows the difference between the Selection List menu on a PC and an iPad.



Aside from the different appearance of the component toolbars, dashboards on mobile devices have certain limitations as far as component behavior:

- The Visual Composer is not available for creating dashboards.
- The Chart Editor is not available for making modifications to a Chart binding.
- The Crosstab Editor is not available for making modifications to a Crosstab binding.
- The Chart ‘Format’ and ‘Properties’ dialog boxes are not available.
- The Chart right-click menu options (‘Hide Legend’, etc.) are not available.

Keyboard Navigation

There are several different techniques by which you can navigate a dashboard using the keyboard (or equivalent voice commands). The table below lists these essential techniques.

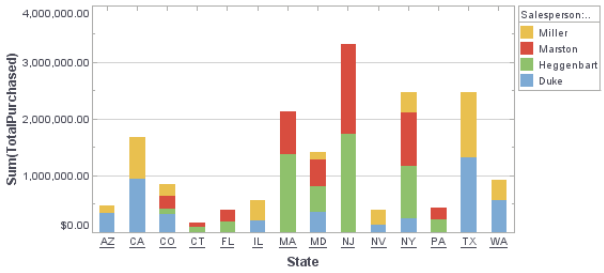
OPERATION	TECHNIQUE
Move focus from one component to another	Press ‘Tab’ or ‘Shift-Tab’
Move focus within a component (e.g., from Range Slider handle to Range Slider body)	Press ‘Up-arrow’ or ‘Down-arrow’
Adjust a component setting (e.g., adjust Range Slider range)	Press ‘Left-arrow’ or ‘Right-arrow’
Make a selection (e.g., choose an item in a Selection List)	Press ‘Space bar’
Move focus within component toolbar	Press ‘Left-arrow’ or ‘Right-arrow’
Move focus from component body to component toolbar	Press ‘Up-arrow’ (repeatedly, if necessary)
Move focus from component toolbar to component body	Press ‘Down-arrow’

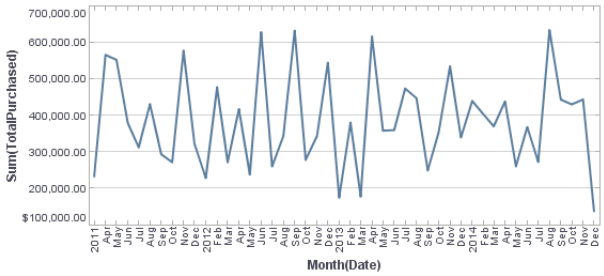
Interacting with Charts

Charts offer several interactive features including brushing, zooming, and filtering. The following sections introduce these features.

Brushing a Chart

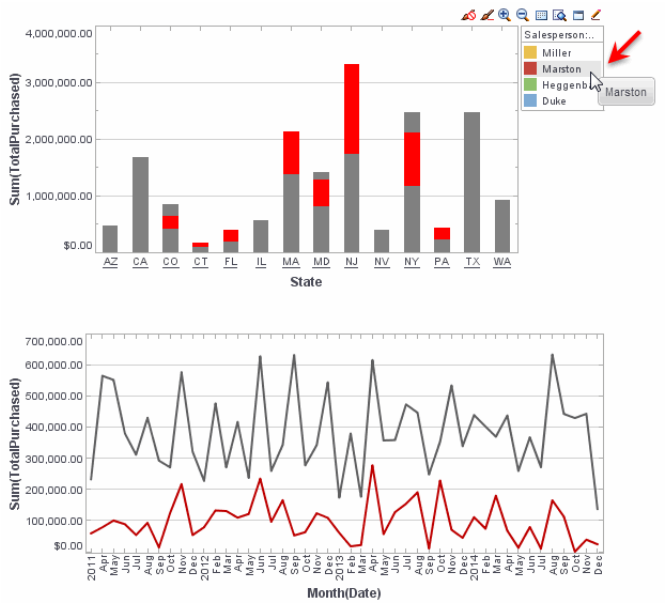
Brushing is a powerful tool that helps you to quickly detect and understand relationships your data. Consider the following charts. (See *Adding a Chart* for instructions on how to create a chart.)





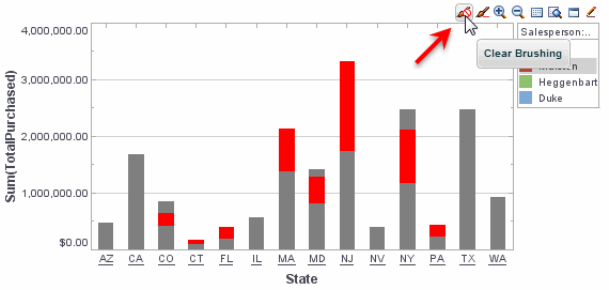
Use brushing to answer the following question: “In what month did salesperson Sue Marston have peak sales?”

1. Double-click the ‘Marston’ label in the legend to brush Sue Marston’s sales in both charts.



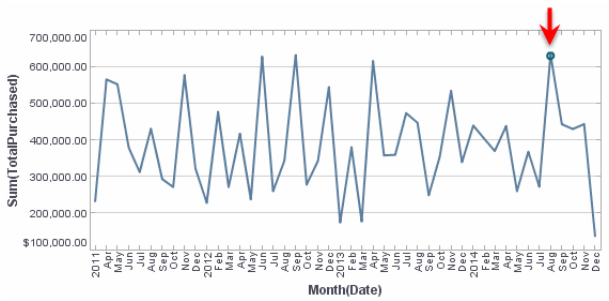
It is now obvious that Marston's greatest sales month was April of 2013, when she sold over $400,000.

2. Press the 'Clear Brushing' button to return to the normal view.



Overall sales were high in August 2014. Use brushing to answer the following question: "Which state saw the greatest sales during this month?"

1. In the second chart, click to select the data point for August 2014.



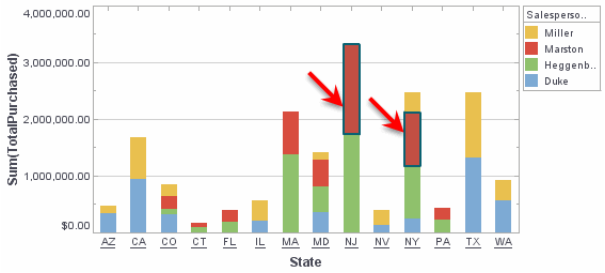
2. Press the ‘Brush’ button on the chart toolbar to highlight all the data corresponding to this month.



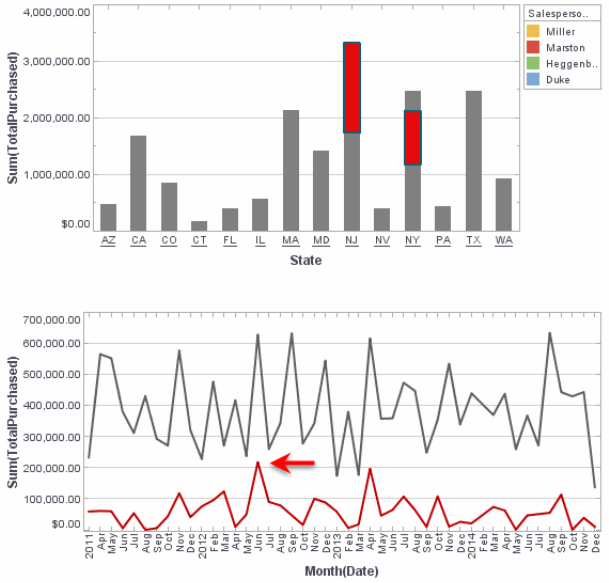
It is clear that Texas had the majority of sales in August 2014.

A majority of Sue Marston’s sales are in New Jersey and New York. Use brushing answer the following question: “During which month were Marston’s combined sales in NJ and NY the greatest?”

1. In the first chart, Ctrl-click to select Marston’s sales both in NJ and NY.



2. Press the ‘Brush’ button on the chart toolbar to highlight all of Marston’s sales corresponding to these states.



It is clear that Marston's sales in NJ and NY reached a peak in June 2012.

See Also

Brushing a Chart, in *Advanced Dashboard Design*, for further details about brushing.

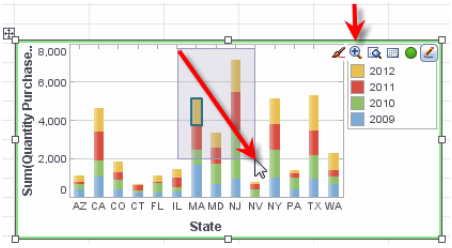
Zooming a Chart

To zoom in on a particular region of a chart, follow the steps below:

See *Selecting Chart Data* in *Advanced Dashboard Design* for more about data selection.

1. Select the data on the chart that you wish to zoom. (Ctrl-click or drag across multiple groups to select them.)

2. Press the 'Zoom' button in the toolbar. This zooms the chart to display only the data selected.



3. Repeat the above steps to zoom in further.

4. To return to the original chart view, press the 'Clear Zoom' button.

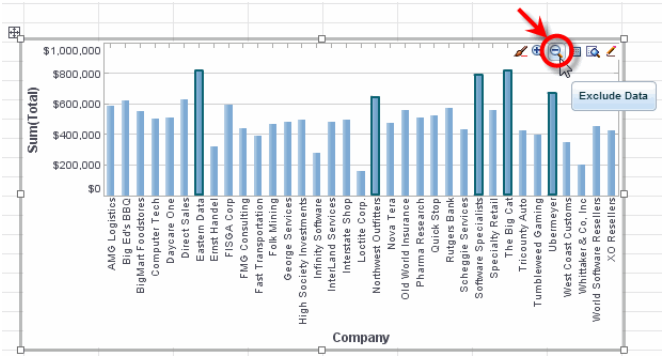


The Zoom feature allows you focus in on a selected set of groups by specifying the groups to *include* in the zoomed view. In some cases, however, you might find it easier to create the desired view by specifying particular groups to *exclude* from the zoomed view.

To exclude particular groups from a chart display, follow the steps below:

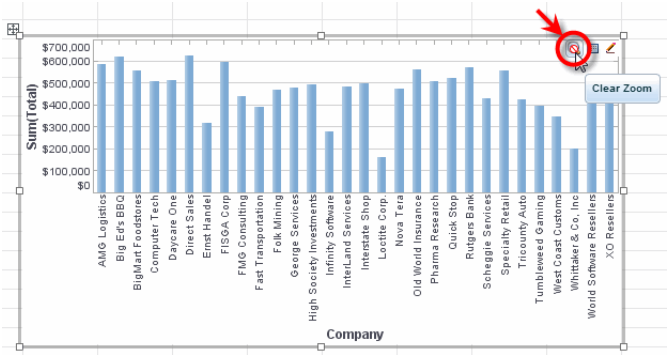
1. Select the data on the chart that you wish to exclude from the zoomed view. (Ctrl-click or drag across multiple groups to select them.)

2. Press the 'Exclude Data' button in the toolbar. This removes the selected data from the chart.



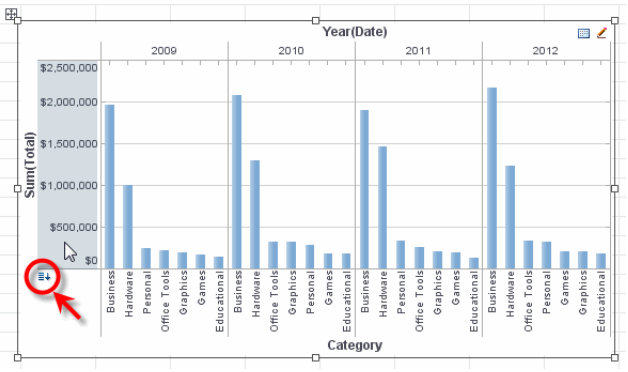
3. Repeat the above steps to exclude further data.

4. To return to the original chart view, press the 'Clear Zoom' button.



Sorting a Chart

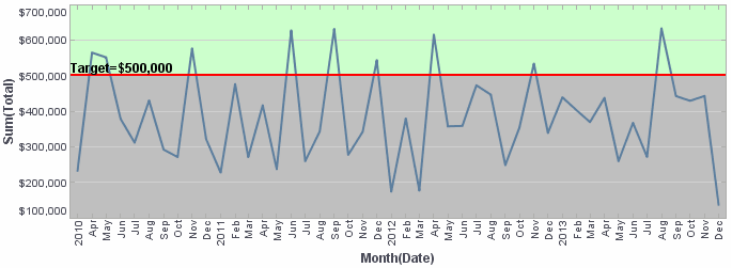
To sort chart groups based on a measure, hover the mouse over the desired measure axis. This displays the 'Sort' button next to the axis. Click the button once to sort the groups in ascending order by measure value, click a second time to sort in descending order by measure value, and click a third time to restore the original order.



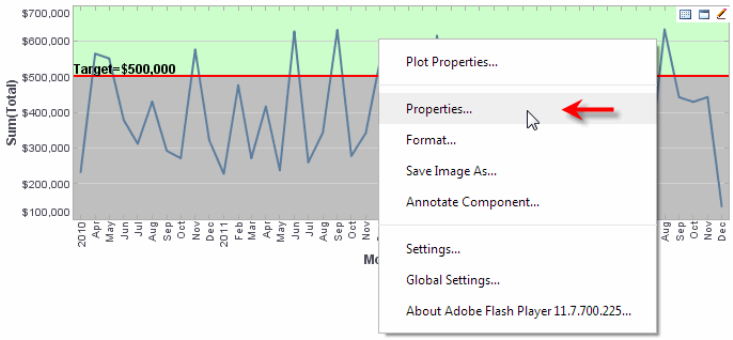
When multiple dimensions are represented on the chart (as in the image above), the 'Sort' button operates on the innermost dimension.

Adding a Target Line

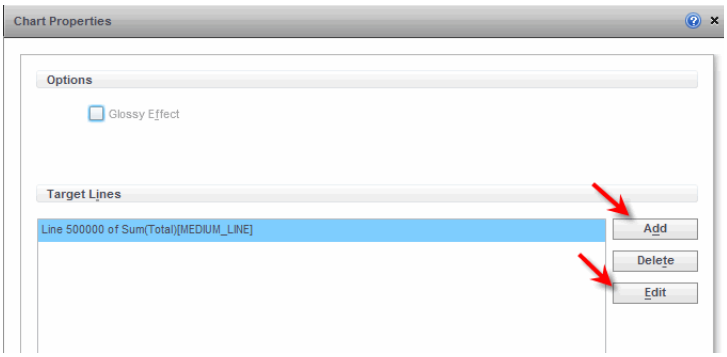
A **target line** is a horizontal or vertical line drawn on the chart that denotes an ideal value (goal or threshold) or representative value (average, minimum, etc.). A **target band** is a horizontal or vertical band drawn on the chart that denotes either an ideal range (e.g., goal zone) or representative range (e.g., span of maximum to minimum). A **statistical measure** is a line or region drawn on the chart to represent one or more statistical quantities derived from the data (confidence intervals, percentiles, etc.).



To create or edit a target line, target band, or statistical measure on a Chart, right-click on the Chart and select 'Properties'. This opens the 'Chart Properties' dialog box.



To create a new target line, band, or statistical measure, press ‘Add’. To edit an existing target line, band, or statistical measure, select the desired target and press ‘Edit’.



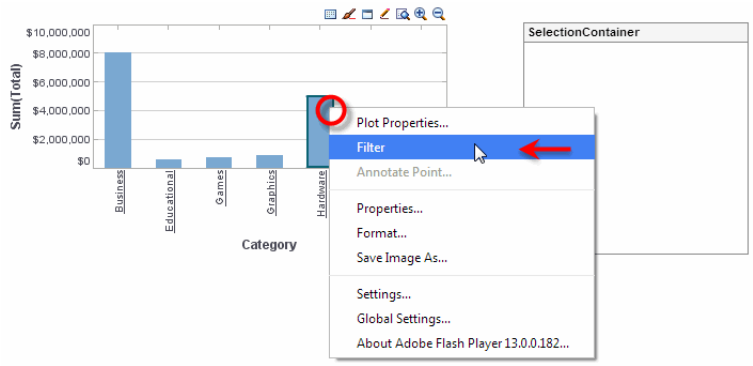
For full information about these chart features, see *Adding a Target Line*, *Adding a Target Band*, and *Adding a Statistical Measure* in *Advanced Dashboard Design*.

Filtering a Chart

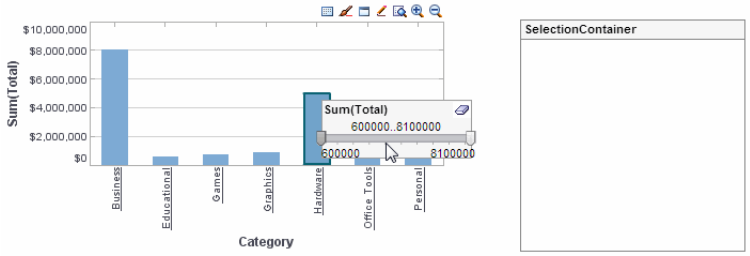
The easiest way to filter a chart is by using Selection components. (See *Filtering with a Selection List* for more information.) However, you can also filter a chart by using the right-click ‘Filter’ option. To do this, follow the steps below:

```
The ‘Filter’ option is only available if the dashboard contains a
Selection Container component.
```

1. Right-click on the label or measure that you want to filter and select ‘Filter’ from the context menu.



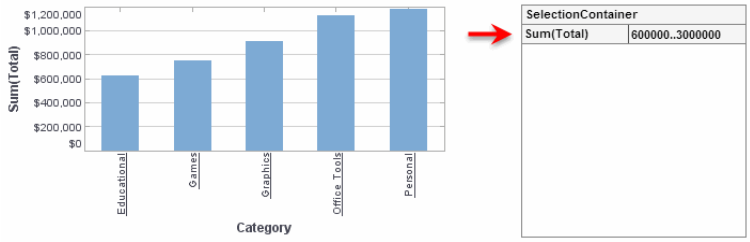
This opens the filtering control for the selected dimension or measure: Selection List for a string-type field, Range Slider for a numerical-type or date-type field.



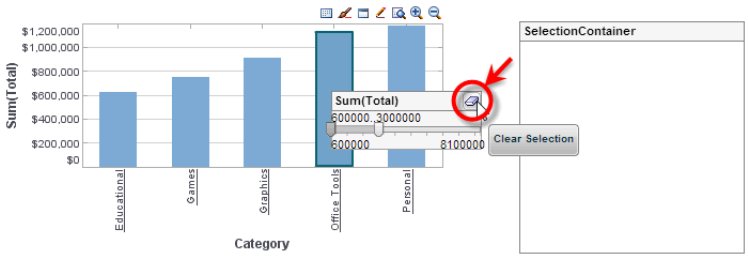
A Range Slider that filters a measure operates on the post-aggregate data displayed in the chart.

2. Make the desired selection using the selection control, and click away to submit. (See *Selection Lists* and *Range Sliders* for information on how to use these controls.)

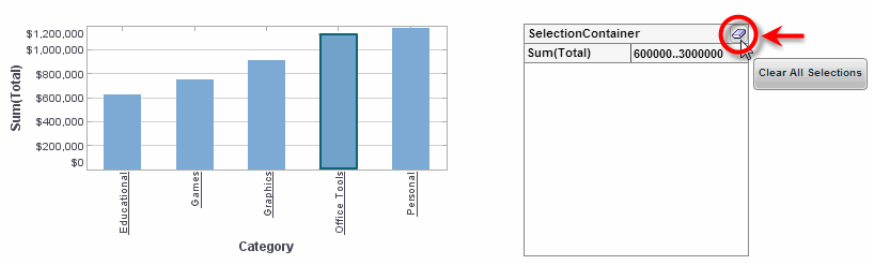
   The selections that you make are shown in the Selection Container.



To remove the filter from a particular field, follow the steps above, and press the 'Clear Selection' button on the selection control.



To remove all filters on the chart, press the ‘Clear Selections’ button in the Selection Container.



See Also

Filtering a Table, for instructions how to filter the aggregated data in a Crosstab or Freehand Table.

Selection Container, for more information about the Selection Container component.

Interacting with Tables

Tables can display a large amount of data in text format. The following sections explain some of the more common table features.

See Also
Tables, in *Advanced Dashboard Design*, for more detailed information.

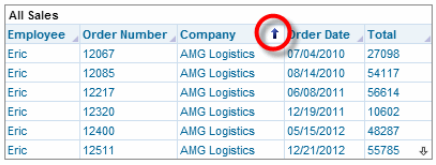
Sorting a Table

Note: Sorting actions for certain tables may be administratively restricted.

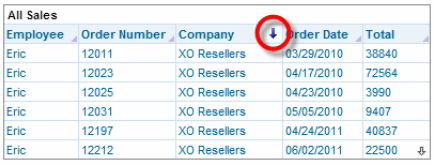
To sort a table column or crosstab column, hover the mouse over the right side of header or data cell and press the 'Sort Column' button.



Press the button once to sort in ascending (alphabetical) order.

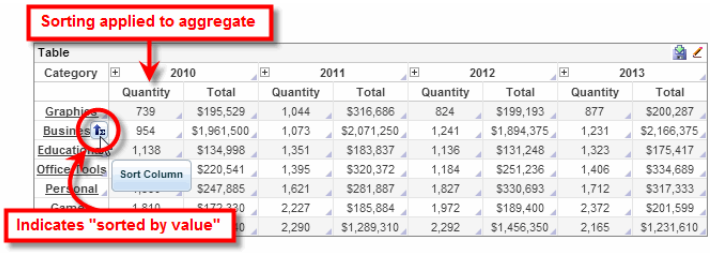


Press the button a second time to sort in descending (reverse-alphabetical) order.



Press the button a third time to restore the original order.

Hover the mouse over a cell to display the current sort order for the field. An up-arrow means "ascending". A down-arrow means "descending". An up-arrow or down-arrow accompanied by a *sigma* indicates that the field is sorted *by aggregate value*. For example, if you sort an aggregated field in ascending order, the sort indicator for the dimension shows that it is sorted in ascending order by the aggregate value.

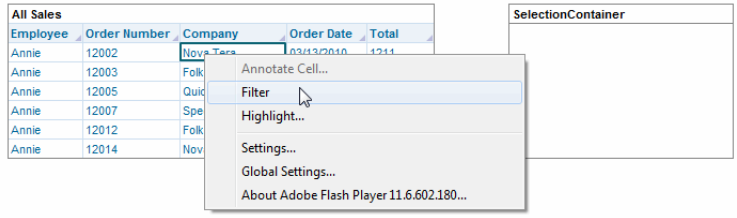


Filtering a Table

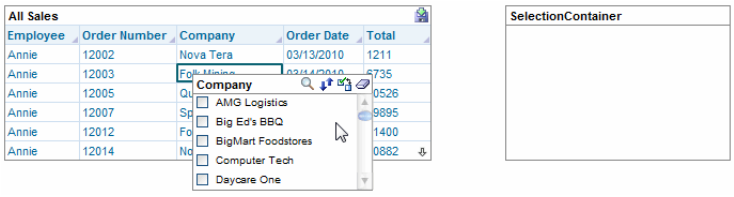
The easiest way to filter a table is by using Selection components. (See *Filtering with a Selection List* for more information.) However, you can also filter a table by using the right-click 'Filter' option. To do this, follow the steps below:

`The 'Filter' option is only available if the dashboard contains a Selection Container component.`

1. Right-click on the column that you want to filter and select 'Filter' from the context menu.



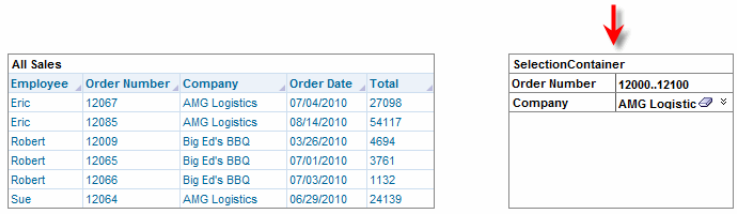
   This opens the filtering control for the column: Selection List for a string-type column, Range Slider for a numerical-type or date-type column.



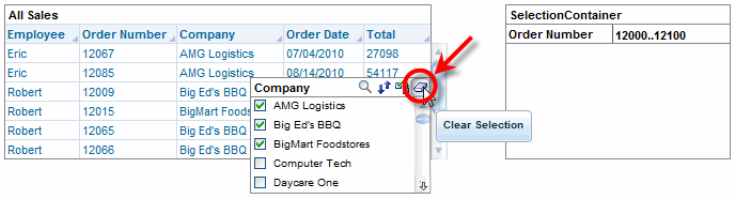
`For a Crosstab or Freehand Table with aggregation, a Range Slider filters the post-aggregate data.`

2. Make the desired selection using the selection control, and click away from the column to submit. (See *Selection Lists* and *Range Sliders* for information on how to use these controls.)

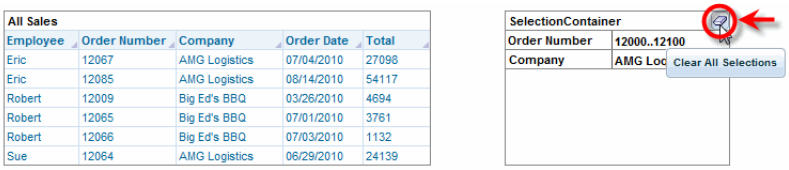
   The selections that you make are shown in the Selection Container.



To remove the filter from a particular column, follow the steps above, and press the 'Clear Selection' button on the selection control.



To remove all filters on the table, press the 'Clear Selections' button in the Selection Container.



See Also

Filtering a Chart, for instructions how to filter the aggregated data in a chart.

Selection Container, for more information about the Selection Container component.

Exporting a Table

To export the data in a Table or Crosstab to Excel, press the 'Export' button in the table title bar.



Modifying Table Data

Regular dashboard tables do not allow you to modify the data in the table cells. However, there are two types of tables that *do* allow you to modify data in table cells:

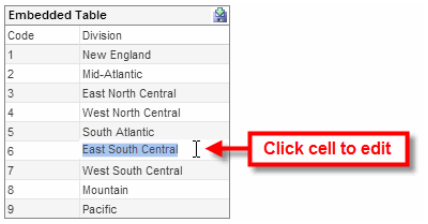
- **Embedded Table**: A table that submits data to a Data Worksheet
- **Editable Table**: A table that submits data to a database

The following sections explain how to modify data in these tables.

Modifying an Embedded Table

An Embedded Table in a dashboard is a table that is based on an Embedded Table in the underlying Data Worksheet. Data that you enter into an Embedded Table is automatically passed to the corresponding Embedded Table in the Data Worksheet.

To enter data into an Embedded Table, simply type values into the desired cells.



Data that you enter is stored with the current bookmark. Therefore, to save the data that you enter, you must save the current bookmark (or create a new bookmark). See *Bookmarking a Dashboard* for more information.

Modifying an Editable Table

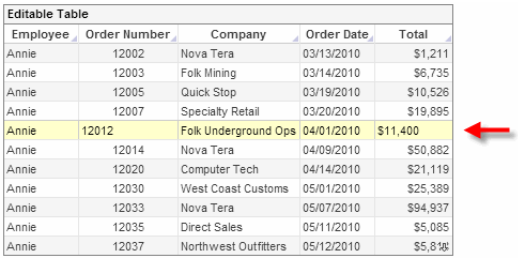
An *editable table* is a table into which you can enter data for submission to the database (database write-back). The dashboard will usually provide a 'Submit' button to allow you to submit your changes.

```
Note: Some columns may not be editable.
```

To enter data into an editable table, simply type values into the desired cells.



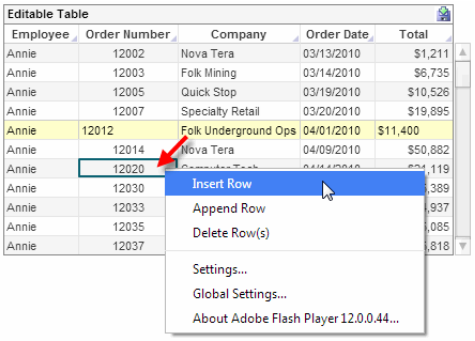
Rows of the table that you have modified are highlighted as shown below.



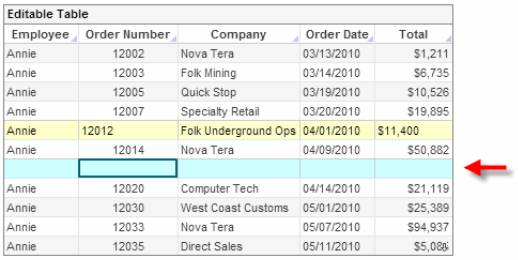
Editable Table

Employee	Order Number	Company	Order Date	Total
Annie	12002	Nova Tera	03/13/2010	$1,211
Annie	12003	Folk Mining	03/14/2010	$6,735
Annie	12005	Quick Stop	03/19/2010	$10,526
Annie	12007	Specialty Retail	03/20/2010	$19,895
Annie	12012	Folk Underground Ops	04/01/2010	$11,400
Annie	12014	Nova Tera	04/09/2010	$50,882
Annie	12020	Computer Tech	04/14/2010	$21,119
Annie	12030	West Coast Customs	05/01/2010	$25,389
Annie	12033	Nova Tera	05/07/2010	$94,937
Annie	12035	Direct Sales	05/11/2010	$5,085
Annie	12037	Northwest Outfitters	05/12/2010	$5,818

To insert a new row before an existing row, right-click the desired row and select ‘Insert Row’ from the context menu. To insert a new row following an existing row, right-click the desired row and select ‘Append Row’ from the context menu.



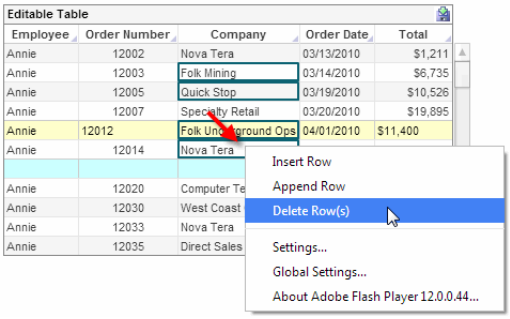
Inserted rows are highlighted as shown below. You can enter text into any cell that appears within an editable column.



Editable Table

Employee	Order Number	Company	Order Date	Total
Annie	12002	Nova Tera	03/13/2010	$1,211
Annie	12003	Folk Mining	03/14/2010	$6,735
Annie	12005	Quick Stop	03/19/2010	$10,526
Annie	12007	Specialty Retail	03/20/2010	$19,895
Annie	12012	Folk Underground Ops	04/01/2010	$11,400
Annie	12014	Nova Tera	04/09/2010	$50,882
Annie	12020	Computer Tech	04/14/2010	$21,119
Annie	12030	West Coast Customs	05/01/2010	$25,389
Annie	12033	Nova Tera	05/07/2010	$94,937
Annie	12035	Direct Sales	05/11/2010	$5,085

To delete an existing row, right-click the desired row and select ‘Delete Row(s)’ from the context menu. To simultaneously delete multiple rows, Ctrl-click or Shift-click to select the desired rows, then right-click on one of the selected cells and choose ‘Delete Row(s)’.


Editable Table
Employee
Order Number
Company
Order Date
Total
Annie 12002 Nova Tera 03/13/2010 $1,211
Annie 12003 Folk Mining 03/14/2010 $6,735
Annie 12005 Quick Stop 03/19/2010 $10,526
Annie 12007 Specialty Retail 03/20/2010 $19,895
Annie 12012 Folk Underground Ops 04/01/2010 $11,400
Annie 12014 Nova Tera
Annie 12020 Computer Te
Annie 12030 West Coast
Annie 12033 Nova Tera
Annie 12035 Direct Sales
Insert Row
Append Row
Delete Row(s)
Settings...
Global Settings...
About Adobe Flash Player 12.0.0.44...

Interacting with Crosstabs

A Crosstab (also called a pivot table) is the tabular equivalent of a chart, displaying aggregated data along one or more dimensions. The following sections explain some basic features of Crosstabs.

See Also
Crosstabs, in *Advanced Dashboard Design*, for more detailed information.

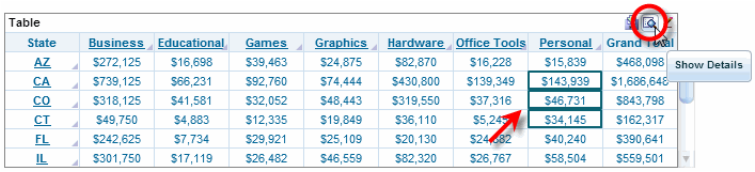
Drilling Down in Crosstab Data

To drill-down into the data in a Crosstab table, follow the steps below:

1. Select the aggregated data into which you want to drill. (You can select data in a Crosstab by clicking to select individual cells, Shift-clicking to select contiguous ranges of cells, and Ctrl-clicking to select discontiguous ranges of cells.)



2. Press the 'Show Details' button in the title bar.



State	Business	Educational	Games	Graphics	Hardware	Office Tools	Personal	Grand Total
AZ	$272,125	$16,698	$39,463	$24,875	$82,870	$16,228	$15,839	$468,098
CA	$739,125	$66,231	$92,760	$74,444	$430,800	$139,349	$143,939	$1,686,648
CO	$318,125	$41,581	$32,052	$48,443	$319,550	$37,316	$46,731	$843,798
CT	$49,750	$4,883	$12,335	$19,849	$36,110	$5,247	$34,145	$162,317
FL	$242,625	$7,734	$29,921	$25,109	$20,130	$24,282	$40,240	$390,641
IL	$301,750	$17,119	$26,482	$46,559	$82,320	$26,767	$58,504	$559,501

This displays the detail data in a panel at the bottom of the screen. You can continue to select different cells without clicking 'Show Details' again.

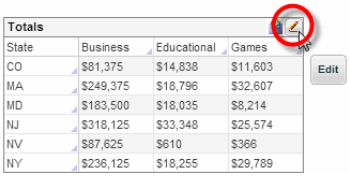
3. To export the data to Excel, press the 'Export' button at the top right.



State	Category	Region	City	Company	Reseller	Date	Paid	Discount	Name	Quantity Pur	Tot
CA	Personal	USA West	Los Angeles	West Coast C	0	12/08/2012	1	0%	True Action	13	$5,174
CA	Personal	USA West	Los Angeles	West Coast C	0	10/28/2012	1	0%	True Action	4	$1,592
CA	Personal	USA West	Los Angeles	West Coast C	0	8/30/2011	1	0%	True Action	19	$7,562
CA	Personal	USA West	Albany	Whittaker & C	0	2/17/2013	1	0%	True Action	20	$7,960
CA	Personal	USA West	Albany	Whittaker & C	0	7/13/2012	1	0%	True Action	18	$7,164
CA	Personal	USA West	Albany	Whittaker & C	0	5/04/2012	1	10%	True Action	1	$398
CA	Personal	USA West	Albany	Whittaker & C	0	2/17/2012	1	0%	True Action	11	$4,378
CA	Personal	USA West	Albany	Whittaker & C	0	6/06/2010	1	0%	True Action	11	$4,378
CA	Personal	USA West	Los Angeles	Computer Tec	1	10/09/2013	1	0%	True Action	18	$7,164
CA	Personal	USA West	Los Angeles	Computer Tec	1	4/14/2010	1	0%	True Action	15	$5,970
CA	Personal	USA West	San Francisc	Direct Sales	1	10/15/2013	0	0%	True Action	4	$1,592
CA	Personal	USA West	San Francisc	Direct Sales	1	10/15/2013	1	0%	True Action	18	$7,164
CA	Personal	USA West	San Francisc	Direct Sales	1	6/17/2012	1	0%	True Action	4	$1,592
CA	Personal	USA West	San Francisc	Direct Sales	1	11/20/2010	1	0%	True Action	9	$3,582

Editing a Crosstab

Press the 'Edit' button in the Crosstab title bar to open the Crosstab Editor to modify the table.



See *Crosstabs* in the *Advanced Dashboard Design* for more information. If the 'Edit' button is not visible in the title bar, this means that the Crosstab is not editable.

Exporting a Crosstab

To export the data in a Table or Crosstab to Excel, press the 'Export' button in the table title bar.



Collaborating

Dashboards support a number of features that make collaboration natural and easy. The following sections explain these features.

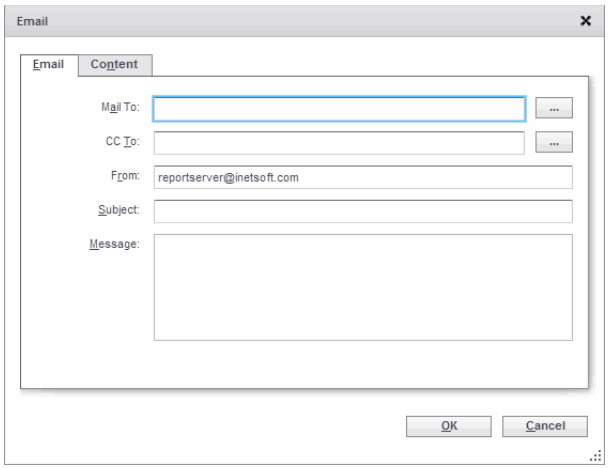
Emailing a Dashboard

To email a dashboard to other users, follow the steps below:

1. Open the dashboard for viewing. (See *Opening a Dashboard*.)
2. Press the 'Email' button on the dashboard toolbar.

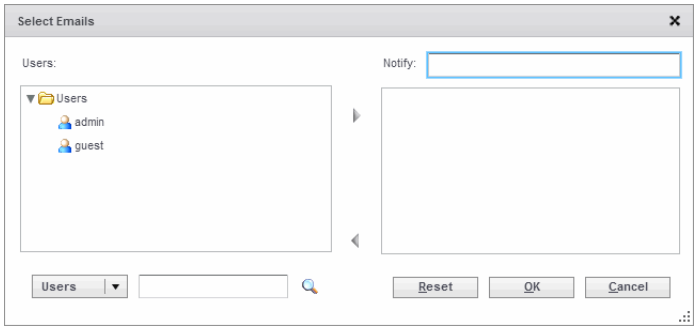


This opens the 'Email' dialog box.



3. On the **Email** tab of the 'Email' dialog box, make the following selections:

   a. In the 'Mail To' field, enter a semicolon-separated list of emails to which the dashboard should be sent. To choose the emails from a list, press the '...' button. This opens the 'Select Emails' dialog box.



Select the users you want to email, and press the ‘Add’ button. When you are finished, press ‘OK’.

b. Repeat the above step to enter the desired emails in the ‘CC To’ field.

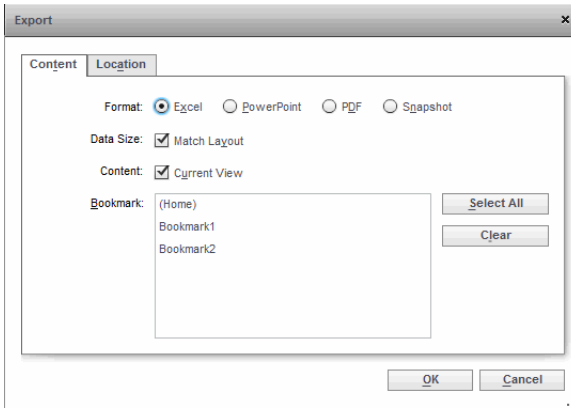
c. In the ‘From’ field, enter the email address that should be used to represent the sender.

d. In the ‘Subject’ field, enter a message for the email subject line.

e. In the ‘Message’ field, enter the body of the email message.

4. Select the **Content** tab, and make the following selections:

a. Choose the export type: ‘Excel’, ‘PowerPoint’, ‘PDF’, or ‘Snapshot’.



See *Importing Assets* in *Advanced Server Administration* for details on incremental deployment.

The ‘Snapshot’ option produces a portable version of the dashboard that incorporates an embedded (static) copy of the underlying data. This “snapshot” dashboard can be loaded into another server installation via incremental deployment or into other InetSoft products such as Style Scope Free Edition (http://www.dashboardfree.com).

b. Select ‘Match Layout’ to obtain the closest possible match between the appearance of the emailed copy and the original dashboard.

c. Select ‘Current View’ to include the current state of the dashboard in the emailed file.

d. Select ‘Include Link’ to include a URL link to the dashboard in the email. The email recipient can click the link to open the dashboard in a new browser tab if they possess the appropriate permissions for that dashboard.

e. Select the bookmarks that you want to include in the exported file. The selected views are converted to individual pages (PowerPoint and PDF) or sheets (Excel) in the exported copy.

5. Press ‘OK’ to close the dialog box and email the dashboard.

Scheduling a Dashboard

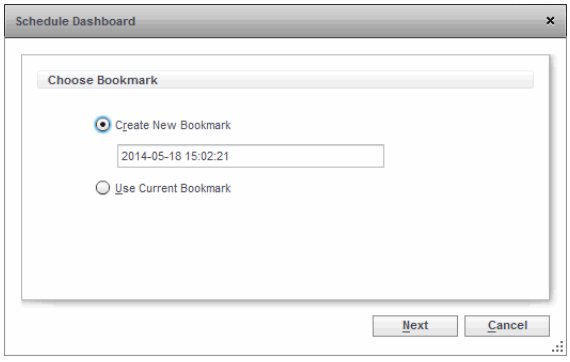
To easily schedule a dashboard for automatic generation and delivery, follow the steps below:

1. Open the dashboard for viewing. (See *Opening a Dashboard*.)



2. Press the ‘Schedule’ button on the toolbar. This opens the ‘Schedule Dashboard’ dialog box.

3. Select ‘Create New Bookmark’ and provide a name for the new bookmark, or select ‘Use Current Bookmark’. (See *Bookmarking a Dashboard* for more information about bookmarks.)

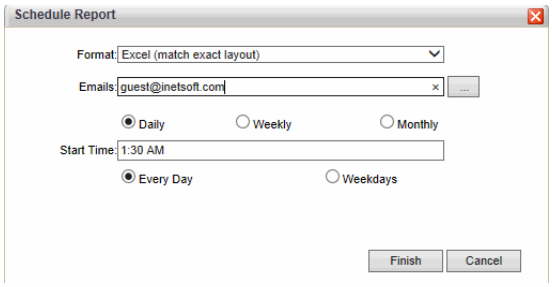


Press ‘Next’.

4. In the ‘Format’ menu, select the format in which the dashboard should be delivered.

5. In the ‘Emails’ field, enter a comma-separated list of email addresses to which the dashboard should be delivered.

6. Select a ‘Daily’, ‘Weekly’, or ‘Monthly’ option and specify the date when the dashboard should be generated.



7. Press ‘Finish’ to close the dialog box.

This creates a new scheduled task that will automatically generate and email the dashboard on the schedule that you specified. The automatically generated dashboard will use the same parameter values that you entered when you originally opened the dashboard.

To view your scheduled tasks or make modifications to a scheduled task, select the **Schedule** tab in the Portal. See *Managing Scheduled Tasks* for more information.

See Also

Scheduling Dashboards and Reports, for another way to schedule a dashboard.

Bookmarking a Dashboard

Bookmarks do not store data, only settings.

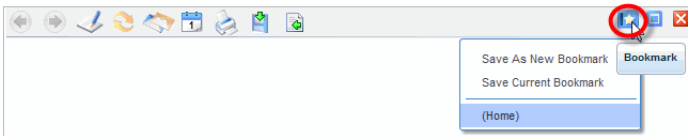
A bookmark allows you save your current dashboard settings (filter selections, menu choices, annotations, etc.) in the User Portal so that you can return to these settings at a later time. The following sections explain how to work with bookmarks.

Saving a New Bookmark

To save a dashboard configuration as a new bookmark in the User Portal, follow the steps below:

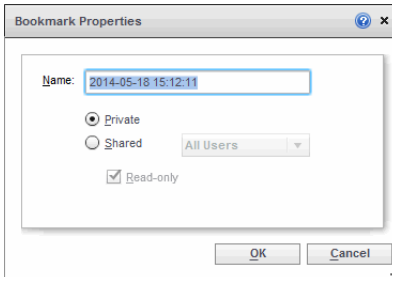
1. Adjust the dashboard settings (filtering selections, menu choices, annotations, etc.) as desired to obtain the configuration that you want to save.

2. Press the ‘Bookmark’ button on the right side of the toolbar. This opens the ‘Bookmark’ menu.



3. Select the ‘Save as New Bookmark’ option. This opens the ‘Bookmark Properties’ dialog box.

4. Enter a name for the new bookmark in the ‘Name’ field. (The default bookmark name is current time.)



5. Select ‘Private’ if you want the bookmark to be visible only to you. Select ‘Shared’ if you want other users of the dashboard to be able to use the bookmark that you have created.

   If you select ‘Shared’, select ‘All Users’ to share the bookmark with all dashboard users or select ‘Same Groups’ to share the bookmark only with users that belong to one or more of your current user groups. (User groups are configured by an administrator.)

6. Enable the ‘Read-only’ option if you do not want other users to modify this bookmark. Otherwise, any user who shares this bookmark will be able to change the bookmark settings.

7. Press ‘OK’ to save the bookmark.

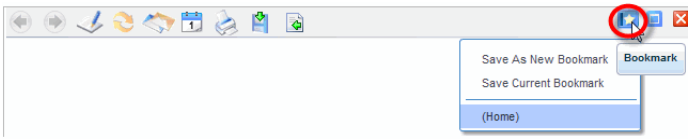
See Also

Annotating a Dashboard, to annotate dashboard components or data.

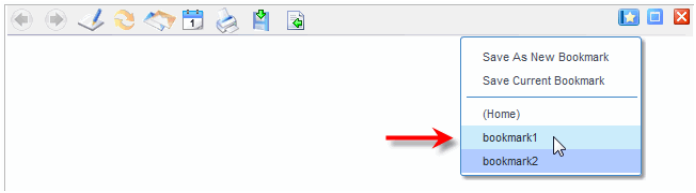
Restoring a Bookmark

To restore a dashboard to the settings saved in a bookmark, follow the steps below:

1. Press the ‘Bookmark’ button on the right side of the toolbar. This opens the ‘Bookmark’ menu.



2. Select the bookmark you want to restore.

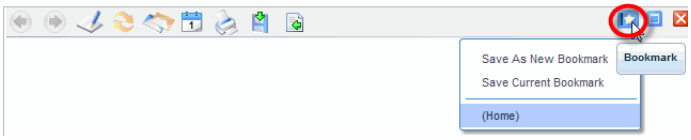


The dashboard is updated with the settings previously saved in the bookmark.

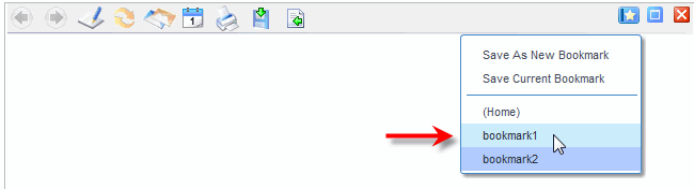
Updating a Bookmark

To update a bookmark with new settings, follow the steps below:

1. Press the 'Bookmark' button on the right side of the toolbar. This opens the 'Bookmark' menu.



2. Select the bookmark you want to update. This bookmark is now the active bookmark.



3. Make any desired changes to the dashboard settings (selections, input components, etc.).

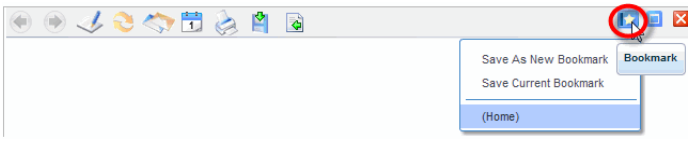
4. Press the 'Bookmark' button again and select 'Save Current Bookmark' or press Ctrl-S on the keyboard. (On Internet Explorer, press Ctrl-Alt-S.) This updates the active bookmark with the current dashboard settings.

Note that you can only update a bookmark for which you have write permission. You can only update the default bookmark (the bookmark shown in italics or the ‘Home’ bookmark) if you have write permission for the dashboard.

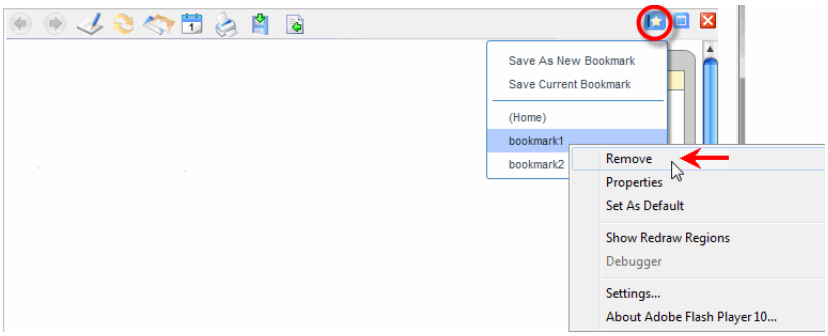
Deleting a Bookmark

To delete a bookmark, follow the steps below:

1. Press the ‘Bookmark’ button on the right side of the toolbar. This opens the ‘Bookmark’ menu.



2. Right-click on the bookmark that you want to delete, and select ‘Remove’ from the context menu.

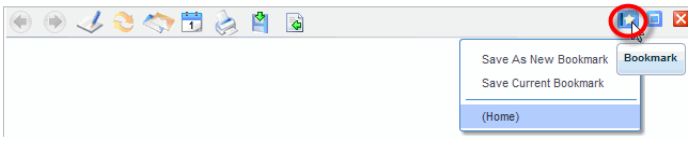


Note that you can only delete a bookmark for which you have write permission. (The ‘Home’ bookmark cannot be deleted.)

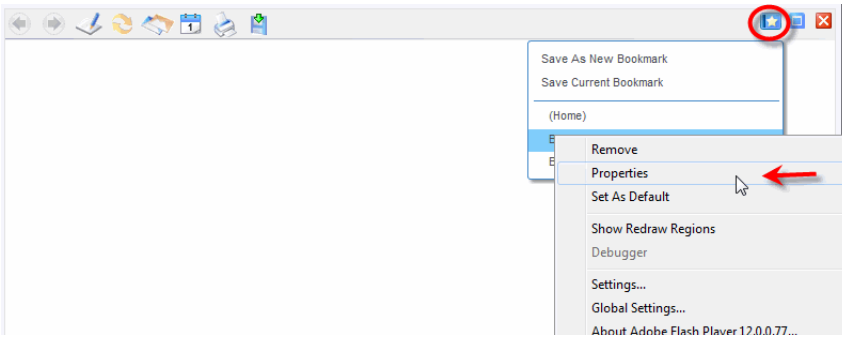
Modifying a Bookmark

To change the name or sharing settings for a bookmark, follow the steps below:

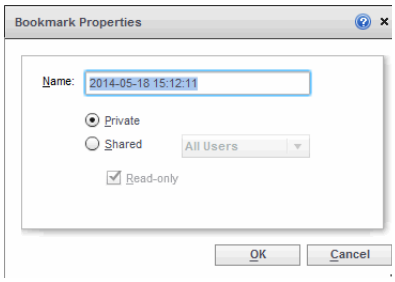
1. Press the ‘Bookmark’ button on the right side of the toolbar. This opens the ‘Bookmark’ menu.



2. Right-click on the bookmark that you want to change, and select ‘Properties’ from the context menu. This opens the ‘Bookmark Properties’ dialog box.



3. Make the desired changes to the bookmark name and sharing. See *Saving a New Bookmark* for more information about these settings.



4. Press ‘OK’ to update the bookmark name and sharing.

Note that you can only modify a bookmark that you have created yourself. (The ‘Home’ bookmark cannot be modified.)

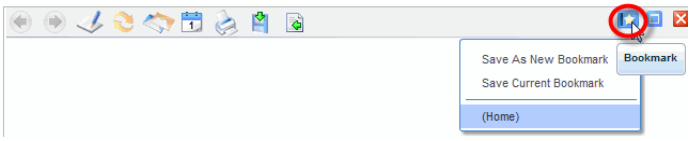
Setting a Default Bookmark

The *default bookmark* for a dashboard specifies the settings (filter selections, menu choices, annotations) that will be in effect each time you re-open the dashboard in the Portal, i.e., the “starting state” of the dashboard.

If you have not yet bookmarked the desired dashboard state, see *Saving a New Bookmark*.

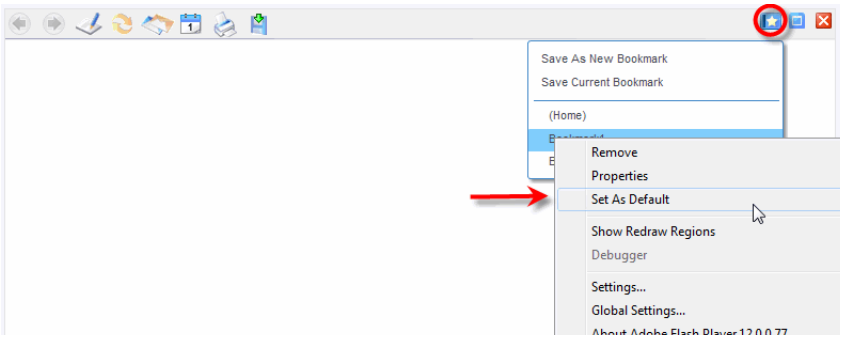
To set an existing bookmark as the default bookmark, follow the steps below:

1. Press the ‘Bookmark’ button on the right side of the toolbar. This opens the ‘Bookmark’ menu.



The default bookmark is subsequently shown in italics.

2. Right-click on the bookmark that you want to set as the dashboard's starting state, and select 'Set As Default'.



The dashboard will now open with the settings defined by this default bookmark.

If you do not specify a default bookmark for a dashboard, the dashboard will open with the settings defined by the 'Home' bookmark. To remove the *default* designation from a bookmark, simply designate a different user-created bookmark or the 'Home' bookmark as the default.

Annotating a Dashboard

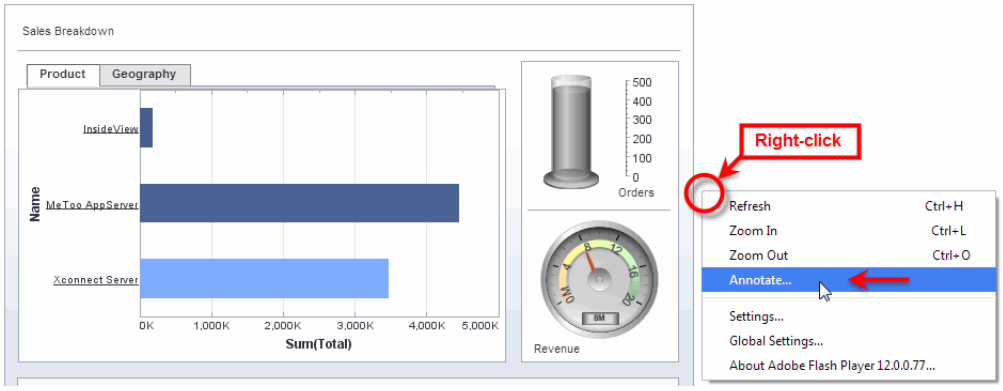
You can add annotations to a dashboard, to an individual dashboard component, and even to an individual data point. The following sections explain how to do this.

Annotating the Dashboard Body

To add annotations to the body of the dashboard, follow the steps below:

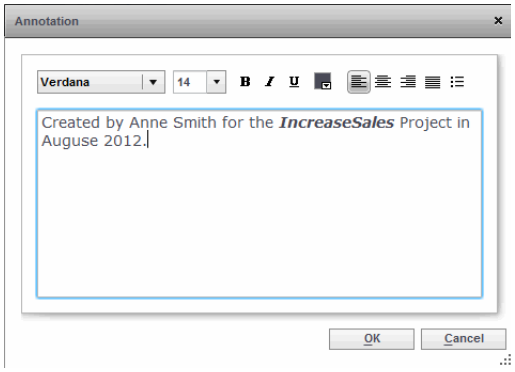
1. Select the bookmark in which you want to save the annotation. (See *Restoring a Bookmark* for information on selecting a bookmark.)

2. Right-click in an empty region of the dashboard, and select 'Annotate' from the context menu.

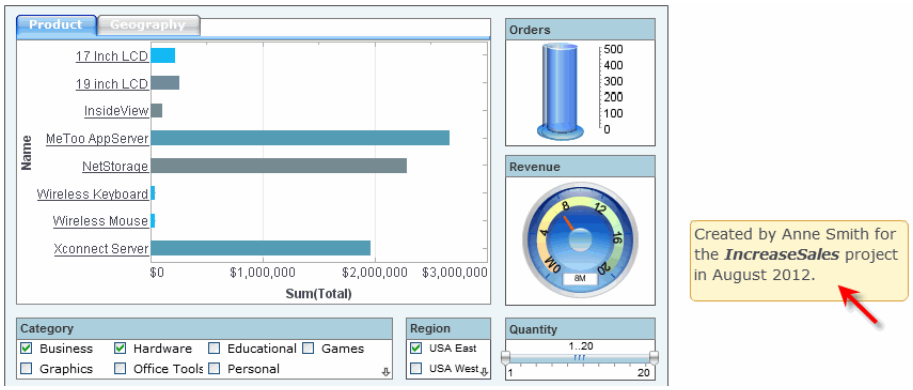


This opens the annotation in the Annotation Editor.

3. Enter the desired annotation text into the Editor, and use the toolbar to visually format the text. See *Editing Annotation Text* for more information on how to use the Editor.



4. Press ‘OK’ to close the Annotation Editor. This adds the annotation to the dashboard.



5. Position the annotation as desired. See *Positioning an Annotation* for more information.

Annotations are always saved together with a bookmark.

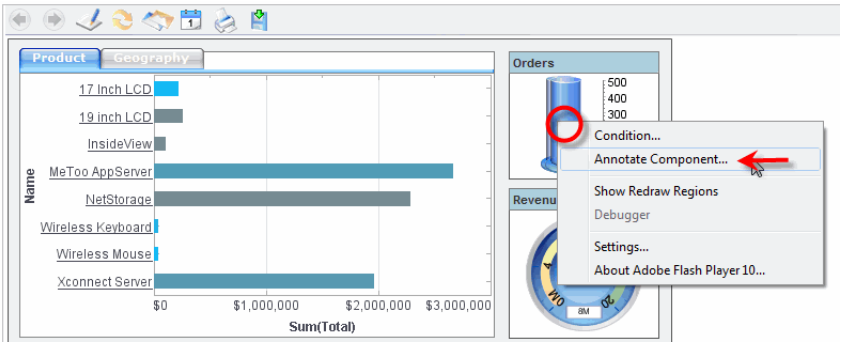
6. To save the annotation, save the current bookmark. (See *Updating a Bookmark* for more information.)

Annotating a Component

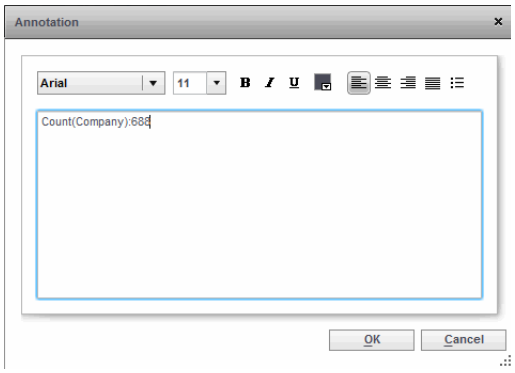
To add an annotation to one of the Data View components (Table, Chart, Crosstab), Output components (Gauge, Image, Text, etc.), or Shape components (Rectangle, etc.) follow the steps below:

1. Select the bookmark in which you want to save the annotation. (See *Restoring a Bookmark* for information on selecting a bookmark.)

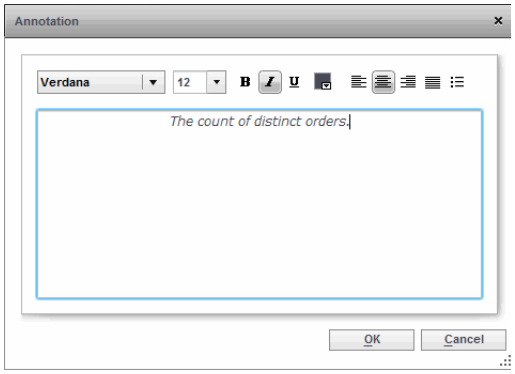
2. Right-click on the Data View, Output, or Shape component, and select 'Annotate Component' from the context menu.



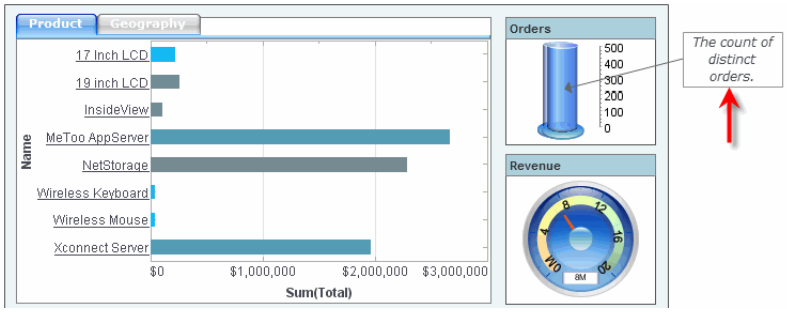
This opens the Annotation Editor. (For an Output component, the Editor displays a default annotation giving the component's current aggregate.)



3. Enter the desired annotation text into the Editor, and use the toolbar to visually format the text. See *Editing Annotation Text* for more information on how to use the Editor.



4. Press ‘OK’ to close the Annotation Editor. This adds the annotation to the dashboard and attaches a callout line to the designated component.



5. Position the annotation as desired. See *Positioning an Annotation* for more information.

Annotations are always saved together with a bookmark.

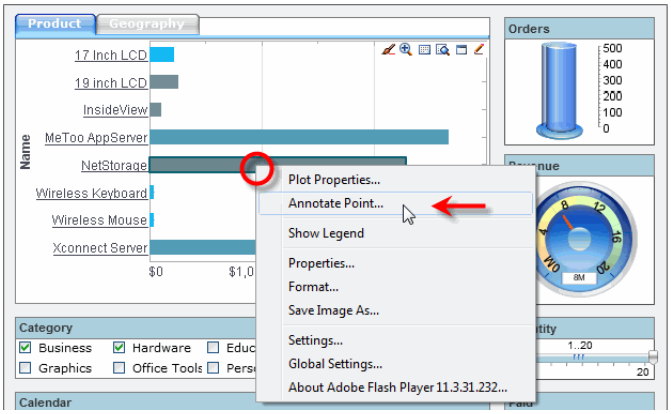
6. To save the annotation, save the current bookmark. (See *Updating a Bookmark* for more information.)

Annotating a Chart Data Point

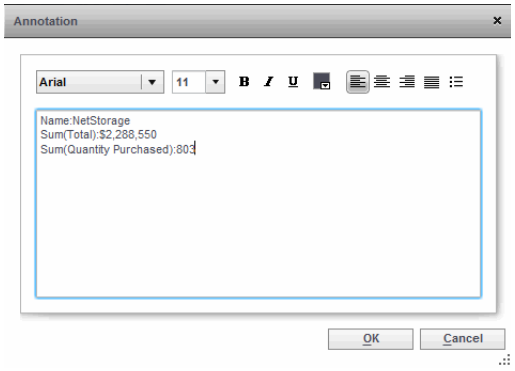
To add an annotation to a data point on a chart, follow the steps below:

1. Select the bookmark in which you want to save the annotation. (See *Restoring a Bookmark* for information on selecting a bookmark.)

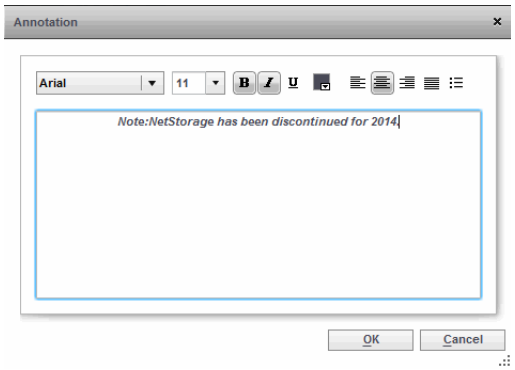
2. Right-click on the chart data point you wish to annotate, and select ‘Annotate Point’ from the context menu.



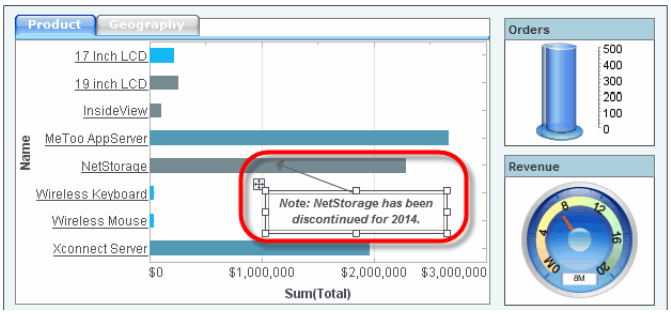
This opens the Annotation Editor, which by default displays the contents of the current data point tooltip.



3. Enter the desired annotation text into the Editor, and use the toolbar to visually format the text. See *Editing Annotation Text* for more information on how to use the Editor.



4. Press ‘OK’ to close the Annotation Editor. This adds the annotation to the dashboard and attaches a callout line to the designated data point.



5. Position the annotation as desired. See *Positioning an Annotation* for more information.

Annotations are always saved together with a bookmark.

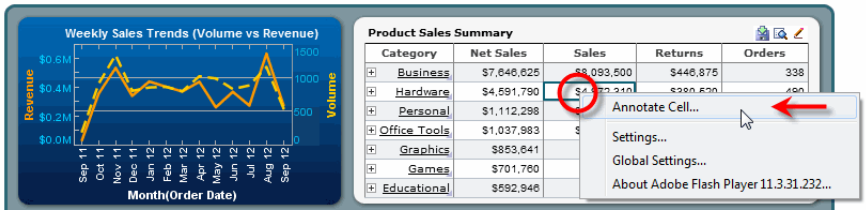
6. To save the annotation, save the current bookmark. (See *Updating a Bookmark* for more information.)

Annotating a Table Cell

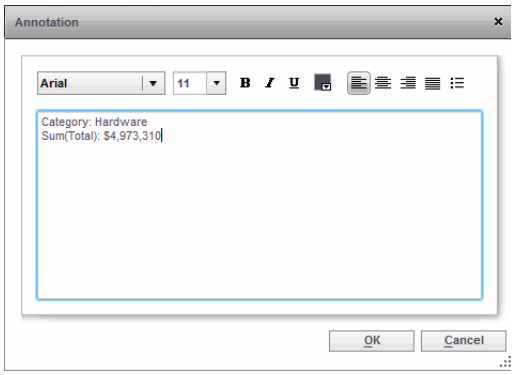
To add an annotation to a data cell in a table, follow the steps below:

1. Select the bookmark in which you want to save the annotation. (See *Restoring a Bookmark* for information on selecting a bookmark.)

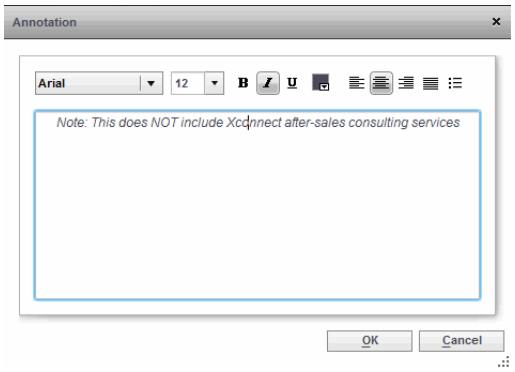
2. Right-click on the table cell you wish to annotate, and select 'Annotate Cell' from the context menu.



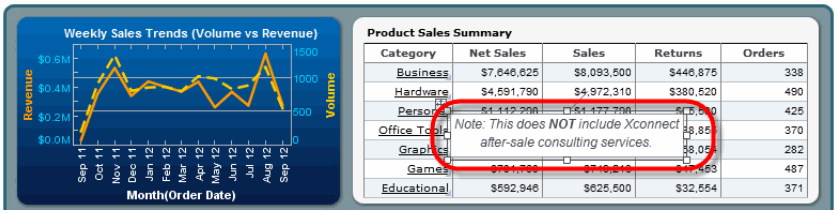
This opens the Annotation Editor, which by default displays the contents of the current cell's tooltip.



3. Enter the desired annotation text into the Editor, and use the toolbar to visually format the text. See *Editing Annotation Text* for more information on how to use the Editor.



4. Press 'OK' to close the Annotation Editor. This adds the annotation to the dashboard and attaches a callout line to the designated table cell.



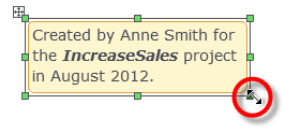
5. Position the annotation as desired. See *Positioning an Annotation* for more information.

Annotations are always saved together with a bookmark.

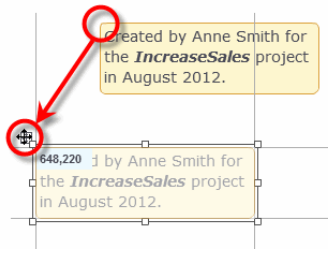
6. To save the annotation, save the current bookmark. (See *Updating a Bookmark* for more information.)

Positioning an Annotation

To resize an annotation, click-and-drag on one of the annotation "resize" handles.

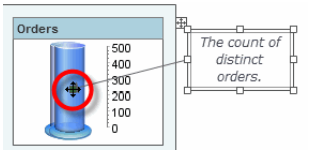


To move the annotation, click-and-drag on the "move" handle, or click and drag on the annotation body.



To format the annotation box or callout line, see *Formating an Annotation*.

To move the callout arrow for a component annotation, click-and-drag on the arrow handle. The callout arrow must be placed within the body of the component.



The callout arrow for a data point annotation cannot be repositioned.

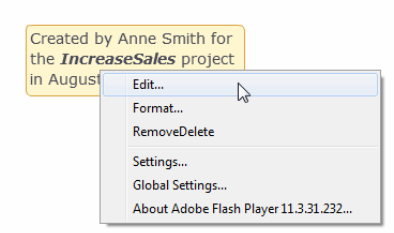
Annotations are always saved together with a bookmark.

To save the annotation after you make changes, save the current bookmark. (See *Updating a Bookmark* for more information.)

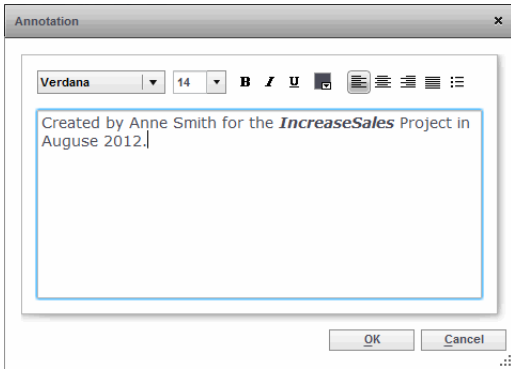
Editing Annotation Text

To edit an existing annotation, first make sure the annotation is visible. (See *Hiding an Annotation*.) Then follow the steps below:

1. Right-click on the annotation and select ‘Edit’ from the context menu.



This open the annotation for editing in the Annotation Editor.



2. Modify the annotation text as desired by typing in the text box.

3. Modify the visual formatting of a block of text as desired by using the formatting tools in the toolbar to adjust the font, size, weight, color, etc.

4. Press 'OK' to close the Editor.

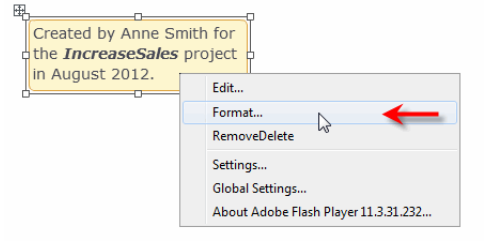
Annotations are always saved together with a bookmark.

5. To save the annotation, save the current bookmark. (See *Updating a Bookmark* for more information.)

Formatting an Annotation

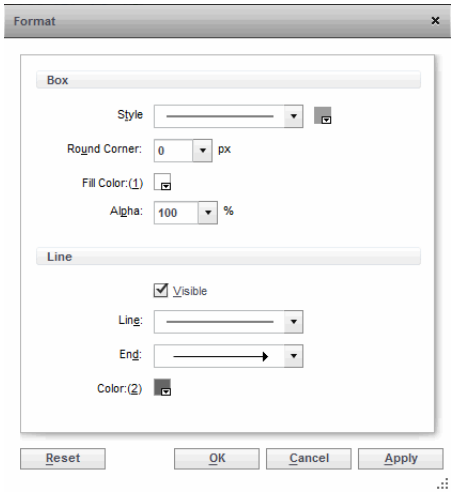
To change the visual appearance of an annotation box or callout line, first make sure the annotation is visible. (See *Hiding an Annotation*.) Then follow the steps below:

1. Right-click on the annotation and select 'Format' from the context menu.



This opens the 'Format' dialog box.

2. Adjust the following settings in the 'Box' panel:



a. From ‘Style’ menu, select a line style for the box border. Select a border color from the adjacent color-picker.

b. From the ‘Round Corner’ menu, select a radius (in pixels) for the box corners. Larger values produce more gently rounded corners, and smaller values produce more sharply rounded corners.

c. From the ‘Fill Color’ menu, select a background color for the annotation box.

d. From the ‘Alpha’ menu, set the background transparency. A value of 0% indicates complete transparency (i.e., fill color not visible), and a value of 100% indicates complete opacity.

3. Adjust the following settings in the ‘Line’ panel:

a. Enable the ‘Visible’ option to display the callout line.

b. From ‘Line’ menu, select a line style for the callout line.

c. From the ‘End’ menu, select the arrow type for the callout line.

d. From the ‘Color’ menu, select a color for the callout line.

4. Press ‘OK’ to close the dialog box.

Annotations are always saved together with a bookmark.

5. To save the annotation, save the current bookmark. (See *Updating a Bookmark* for more information.)

Hiding an Annotation

When a dashboard or component annotation is hidden, it appears on the dashboard or component as a small “note” icon. The toolbar ‘Annotation’

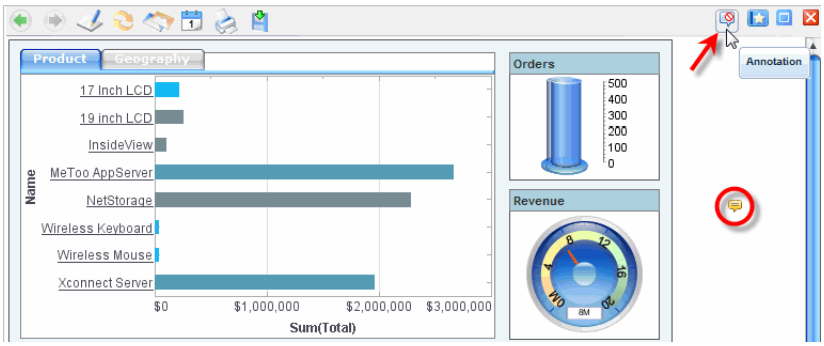
button allows you toggle annotations between their hidden and visible states.

```
Every annotation is associated with a particular bookmark.
```

To display all annotations for a particular bookmark, follow the steps below:

1. Select the bookmark for which you want to see annotations. (See *Restoring a Bookmark* for information on selecting a bookmark.)

2. Press the ‘Annotation’ button in the toolbar.



If you export a dashboard while annotations are visible, the annotations will be included in the export.

To hide all annotations, simply press the ‘Annotation’ button in the toolbar a second time.

Importing/Exporting a Dashboard

You can export an image of an entire dashboard, or export data from a particular dashboard component. The following sections explain how to do this.

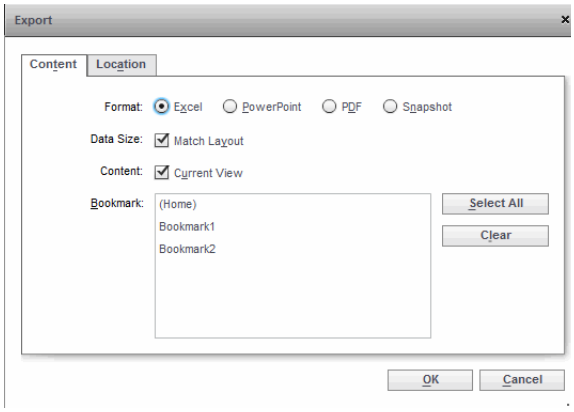
Exporting a Static Dashboard

To export a static copy of the dashboard into Excel, PowerPoint, or PDF format, follow the steps below:

1. Press the ‘Export’ button in the dashboard toolbar. This opens the ‘Export’ dialog box.

2. Select the **Content** tab, and make the following selections:

   a. Choose the export type: ‘Excel’, ‘PowerPoint’, ‘PDF’, or ‘Snapshot’.



See *Importing Assets* in *Advanced Server Administration* for details on incremental deployment.

The 'Snapshot' option produces a portable version of the dashboard that incorporates an embedded (static) copy of the underlying data. This "snapshot" dashboard can be loaded into another server installation via incremental deployment or into other InetSoft products such as Style Scope Free Edition (http://www.dashboardfree.com).

b. Select 'Match Layout' to obtain the closest possible match between the appearance of the exported copy and the original dashboard.

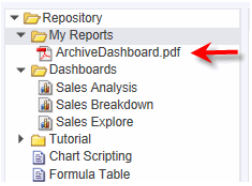
c. Select 'Current View' to include the current state of the dashboard in the exported file.

d. Select the bookmarks that you wish to include in the exported file. The selected views are converted to individual pages (PowerPoint and PDF) or sheets (Excel) in the exported copy.

```
Note: An 'anonymous' user cannot save to the repository, and a
Snapshot cannot be saved to the repository.
```

3. Select the **Location** tab. Choose 'Desktop' to save the exported file onto the local machine. Choose 'Repository' and select a repository location to save the exported file onto the server.

   When you export a dashboard to the repository, the dashboard is displayed within the repository tree but does not pull live data from the database. The exported dashboard contains archived data, reflecting the state of the dashboard at the time that the export was made.



4. Press ‘OK’ to close the dialog box and proceed with the export.

Exporting an Editable Dashboard

`Note: Offline dashboard editing features are available only if product is licensed for Viewsheet Forms.`

Dashboards provide several features to assist in *offline data editing*. You can export data from Table and Input components into an Excel file, edit the data values and make input selections within the Excel file, and re-import the revised data and selections into the dashboard at a later time. This allows you to edit a dashboard *offline*, while not actively logged into the server.

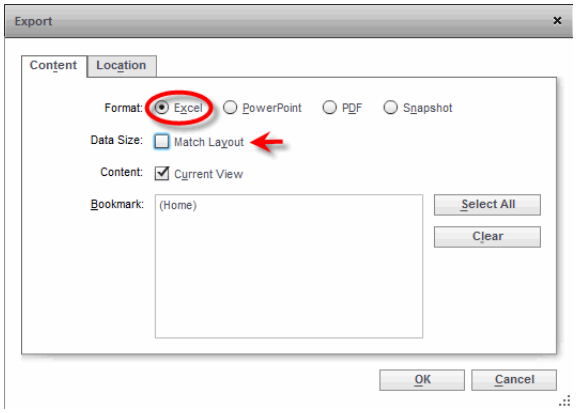
To export a dashboard for editing (and subsequent importing), follow the steps below:

1. Press the ‘Export’ button in the dashboard toolbar. (**Note**: Do not use the ‘Export’ button in the table title bar.)



2. On the **Content** tab, select the ‘Excel’ export option.

3. Deselect the ‘Match Layout’ option.



4. Under the **Location** tab, select the 'Desktop' option.

5. Press 'OK' to close the dialog box and export the dashboard.

This saves the Excel file (with the same name as the dashboard) in the browser's default download directory. You can now open the file in Excel, and edit the tables or modify input elements as desired. To re-import the modified Excel data back into the dashboard, see *Importing Excel Data into a Dashboard*.

See Also

Exporting Table Data for Editing, in *Advanced Dashboard Design*, for information on how to configure editing features.

Importing Excel Data into a Dashboard

```
Note: Offline dashboard editing features are available only if
product is licensed for Viewsheet Forms.
```

Dashboards provide several features to assist in *offline data editing*. You can export data from Table and Input components into an Excel file, edit the data values and make input selections within the Excel file, and re-import the revised data and selections into the dashboard at a later time. This allows you to edit a dashboard *offline*, while not actively logged into the server.

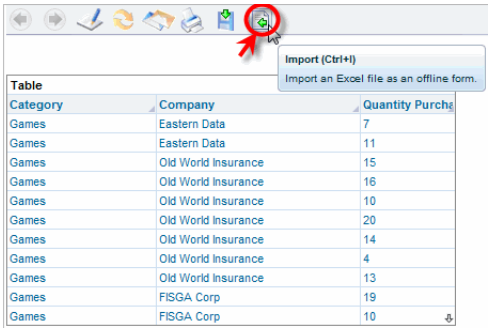
To import data from Excel into a dashboard, the following conditions must be satisfied:

- The Excel file must be created by the procedure described in *Exporting an Editable Dashboard*. (If the Excel file was created by another process, the import will not succeed.)
- Each editable Table and Input component in the dashboard must have the same name (set by the 'Name' field in the 'Properties' dialog box) as the corresponding table or input in the Excel file. These names are used

to match the Excel components with the corresponding dashboard components.

When you have verified the above conditions, follow the steps below to import the modified data from the Excel file:

1. Press the ‘Import’ button in the dashboard toolbar. This opens the file selection dialog box.



2. Choose the Excel file that you want to import, and press ‘Open’.

This uploads the Excel data into the dashboard tables with corresponding names, and updates input elements with revised settings. Records that have been added or edited within Excel are highlighted for better visibility.

See Also

Importing a Data File, for information on importing Excel data into a Data Worksheet.

Committing User-Modified Data to Database (Database Write-Back), in *Advanced Dashboard Scripting*, for information on committing modified table data to a database.

Exporting Table Data for Editing, in *Advanced Dashboard Design*, for information on how to configure editing features.

Using Dashboard Components

The following sections explain how to use the features of individual dashboard components.

Inputs

Note: The dashboard interface operates in a slightly different way on mobile devices such as phones and tablets. See *Using a Dashboard on a Mobile Device* for details.

You can enter values into a dashboard by using Input components. The following Input components are often use to dynamically adjust dashboard properties such as the data displayed on chart axes, and can also be used to modify tables and variables in an underlying Data Worksheet.

Slider: Drag the handle to the desired value.



Spinner: Press the arrow buttons to select the desired value, or type a numerical value directly into the number field.



CheckBox: Click to select the desired item or items.



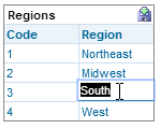
RadioButton: Click to select the desired single item.



ComboBox: Expand the menu and select the desired item. In some cases, you can also manually type values into the ComboBox.



Embedded Table: Type data into the Table component. Click away from the Table to submit the data.



TextInput: Enter text into the box. Often a TextInput component is accompanied by a 'Submit' or 'OK' button to allow you to submit your text.

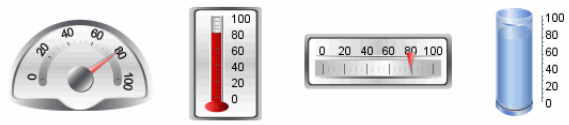


See Also

Adding an Input, for information on how to create an Input control.

Outputs

Output components such as gauges, scales, and thermometers display a single aggregate value, and do not allow input or manipulation.



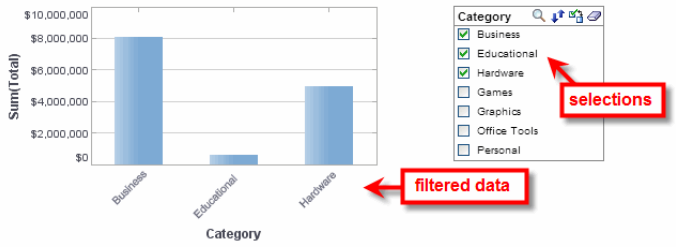
See Also

Output Components, in *Advanced Dashboard Design*, for more detailed information.

Selection Lists

A Selection List filters the data displayed by data view components (Tables, Charts, etc.) and output components (Gauges, Thermometers, etc.). A Selection List can provide either single-selection or multiple-selection capability, and can be configured to submit selections immediately when you make the selection or on-demand when you press the 'Apply' button.

To filter data using a Selection List, select the desired item(s) in the list. If an 'Apply' button is visible in the title bar, press the 'Apply' button to submit your selections.



If there is no ‘Apply’ button in the title bar, each selection you make is submitted immediately. To make multiple selections before submitting, hold down the Ctrl-key on the keyboard while making your selections.

The Selection List provides a mini-toolbar that allows you make convenient modifications to the current selection. Move the mouse over the Selection List to see the mini-toolbar. The following functions are available.

	**Search** Press the ‘Search’ button and enter a search term in the text field. As you type, the Selection List updates to display only items having a match for your search text (appearing anywhere in the item). Press the ‘X’ button in the text field to close the Search feature.
	**Sort Selection** Press the ‘Sort Selection’ button once to sort the items in ascending numerical or alphabetical order. Press the button again to sort in descending numerical or reverse alphabetical order. Press the button a third time to reset the original order.
	**Reverse Selection**. Press the ‘Reverse Selection’ button to deselect all currently selected items and select all currently unselected items.
	**Clear Selection**. Press the ‘Clear Selection’ button to deselect all items.
	**Apply**. Press ‘Apply’ to submit the current selection. If the ‘Apply’ button is not visible, this means that your selections are submitted automatically.

Selection Lists make use of five different icons to designate the status of items in the list.

	**Data selected and included**. The user has explicitly selected this item, and the corresponding records are included in the returned data set.
	**Data selected but not included**. The user has explicitly selected this item, but more recent explicit selections in other selection lists have now made this selection incompatible. The corresponding data is therefore *not* included in the returned data set.
	**Data included but not selected**. The user has not explicitly selected this item, but has made other selections that *implicitly* select this item as well. The corresponding records are included in the returned data set (as a result of those other selections).

	**Data compatible but not included**. The user has not explicitly selected this item, and the item remains *compatible* with existing user selections. This item can therefore be selected to add additional records to the returned data set.
	**Data incompatible**. The user has not explicitly selected this item, and this item is *incompatible* with existing selections. (It appears listed under 'Others'). Selecting this *clears* all other selections and starts fresh.

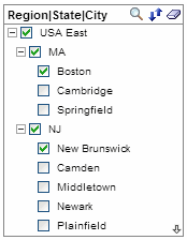
See Also

Filtering with a Selection List, for a complete example of using Selection Lists.

Selection List, in *Advanced Dashboard Design*, for information on designing a Selection List.

Selection Trees

The Selection Tree is a hierarchical version of the Selection List that allows you to select items at multiple levels. See *Selection Lists* for operation of the mini-toolbar and the meaning of the different selection icons.

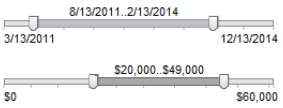


See Also

Selection Tree, in *Advanced Dashboard Design*, for information on designing a Selection Tree.

Range Sliders

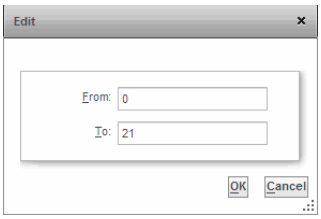
The Range Slider component allows you to select a date range or numerical range.



To adjust the start-point or end-point of a range, drag the left or right handle. To slide the entire range, drag the slider body.

To quickly set the right side of the range to its maximum, double-click on the right handle. To quickly set the left side of the range to its minimum, double-click on the left handle. To quickly set the slider to its full range, double-click on the slider body.

To enter precise values for the start-point or end-point, right-click on the Range Slider and select 'Edit' from the context menu. In the 'Edit' dialog box, enter the desired values and press 'OK'.



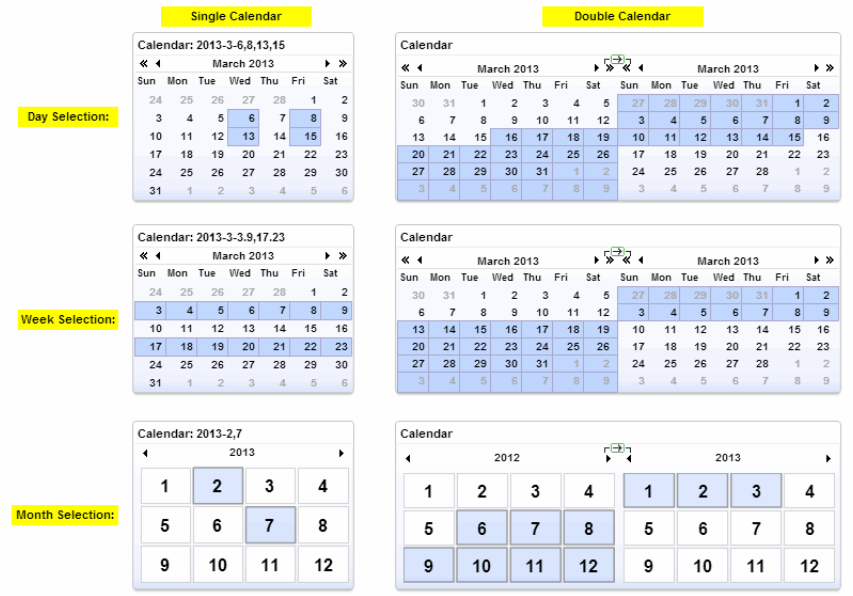
See Also

Range Slider, in *Advanced Dashboard Design*, for information on designing a Range Slider.

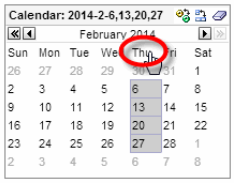
Calendars

The Calendar component provides a calendar interface that allows you to filter data based on a range of dates or a comparison of dates. The Calendar is similar in purpose to a *Range Sliders*, but provides functionality beyond simple range selection.

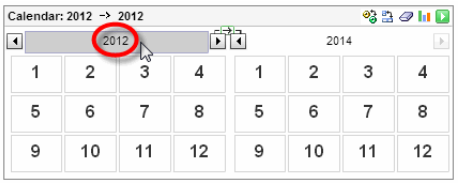
The following illustration shows some of the types of ranges that you can select by clicking on different regions in a Calendar. Note that the Calendar title bar indicates the selected range.



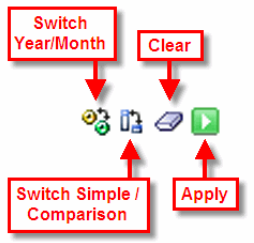
To select a column of days (e.g., all Thursdays in a month), click the name of the desired day.



To select an entire year, click the name of the year.



The Calendar component provides a mini-toolbar in the title bar that allows you make convenient modifications to the calendar. (Move the mouse over the Calendar to see the mini-toolbar.)



The title bar buttons are described below.

	**Switch to Year View** and **Switch to Month View** Press to alternate between year and month styles.
	**Switch to Range/Comparison** and **Switch to Simple View** Press to alternate between single- and double- calendar modes. In single-calendar mode, selections are applied immediately. In double-calendar mode, press the 'Apply' button to submit your selections.
	**Switch to Comparison Mode** and **Switch to Date Range Mode** Press to switch between date range selection and date comparison operations.
	**Clear** Press to remove the specified date information.

	**Apply** Press ‘Apply’ to submit the selection. If the ‘Apply’ button does not appear, this means that your selection is submitted automatically.

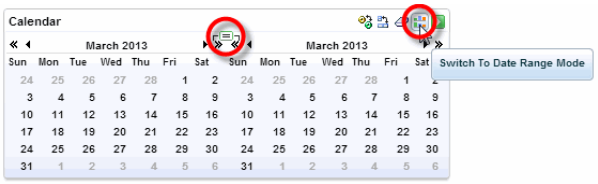
See Also

Calendar, in *Advanced Dashboard Design*, for information on designing a Calendar.

Filtering a Range of Dates

To filter a range of dates using the Calendar component, follow the steps below:

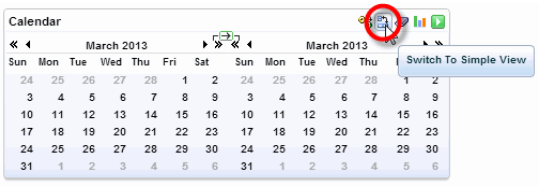
1. If the Calendar is currently in ‘Comparison’ mode (indicated by an “=” symbol between the left and right calendar pages), press the ‘Switch to Date Range Mode’ button, or click the “=” symbol to switch to ‘Date Range’ mode.



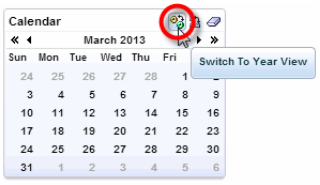
2. (Optional) To switch the Calendar from single-page to double-page display, press the ‘Switch to Range/Comparison’ button.



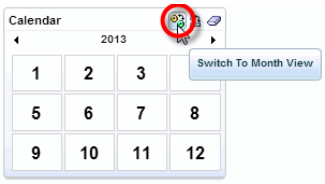
   To switch the Calendar from double-page to single-page display, press the ‘Switch to Simple View’ button.



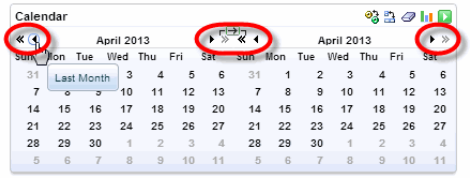
3. (Optional) To switch the Calendar from displaying a single month to displaying the entire year, press the ‘Switch to Year View’ button.



To switch the Calendar from displaying the entire year to displaying a single month, press the ‘Switch to Month View’ button.



4. Press the ‘Last Year’/‘Last Month’ buttons and ‘Next Year’/‘Next Month’ buttons to display the appropriate calendar pages.



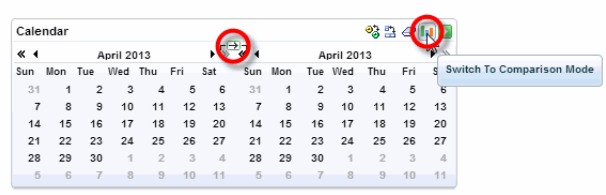
5. Click a date on the Calendar to select the start date of the range. You can select a day, week, or month, depending on the how the Calendar has been configured.

   In a ‘Month View’ calendar, click to select a day or week, or click the name of the month at the top of the calendar to select the entire month. In a ‘Year View’ calendar, click to select a month, or click the year at the top of the calendar to select the entire year.

6. Click another date on the Calendar to select the end date of the range, or Ctrl-click to select discontiguous dates.

7. If the Calendar is double-page view, press the ‘Apply’ button to submit your selection. (If the Calendar is in single-page view, the selection is applied automatically.)

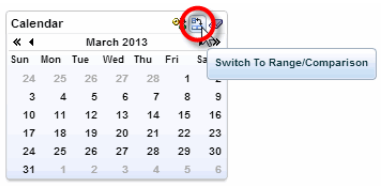
The range that you select is used to filter all data view and output components that are based on the same Data Block as the Calendar component.



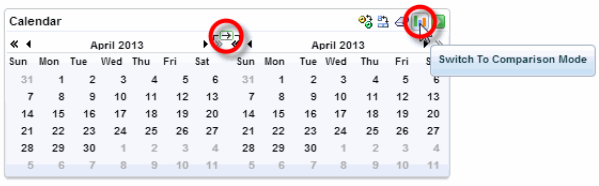
Comparing a Range of Dates

To compare a range of dates using the Calendar component, follow the steps below:

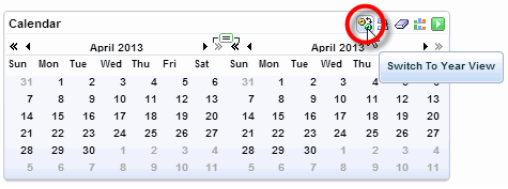
1. If the Calendar is in single-page view, press the ‘Switch to Range/ Comparison’ button to switch to double-page display.



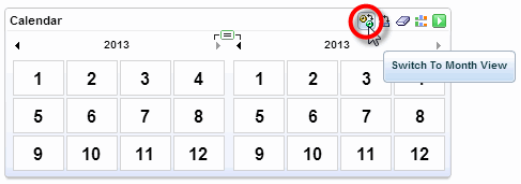
2. If the Calendar is currently in ‘Range’ mode (indicated by an arrow symbol between the left and right calendar pages), press the ‘Switch to Comparison Mode’ button, or click the arrow symbol to switch to ‘Date Comparison’ mode.



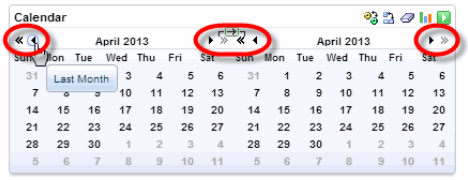
3. (Optional) To switch the Calendar from displaying a single month to displaying the entire year, press the ‘Switch to Year View’ button.



To switch the Calendar from displaying the entire year to displaying a single month, press the ‘Switch to Month View’ button.



4. Press the ‘Last Year’/‘Last Month’ buttons and ‘Next Year’/‘Next Month’ buttons to display the appropriate calendar pages for the comparison that you wish to make.



5. Select a date range to compare on the left page of the Calendar. You can select a day, week, or month, depending on the how the Calendar has been configured.

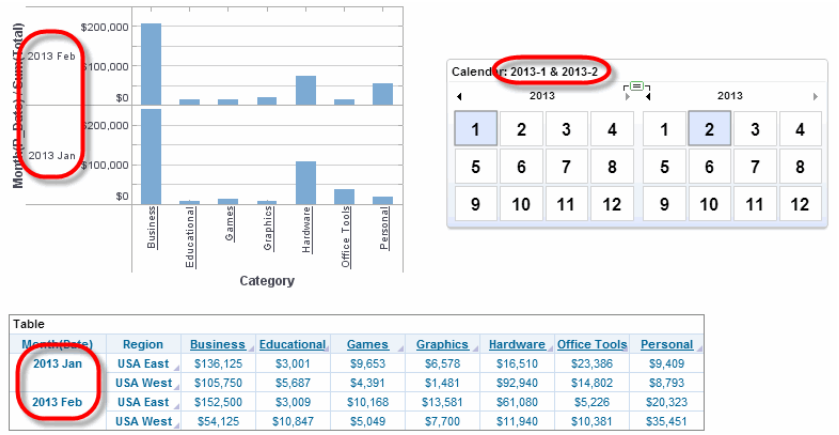
   In a ‘Month View’ calendar, click to select a day or week, or click the name of the month at the top of the calendar to select the entire month. In a ‘Year View’ calendar, click to select a month, or click the name of the year at the top of the calendar to select the entire year. Ctrl-click to select discontiguous dates.

The initial selection that you make will be mirrored on the right page of the Calendar, but you can change this in the next step.

6. Select a date range to compare on the right page of the Calendar. You can select a day, week, or month, depending on the how the Calendar has been configured. Ctrl-click to select discontiguous dates.

7. Press the ‘Apply’ button to submit your selection.

The date ranges that you select are used to generate comparisons in all Charts and Crosstabs that are based on the same Data Block.



Selection Containers

A Selection Container provides convenient access to multiple selection elements, and allows you to add new Selection Components into the dashboard.

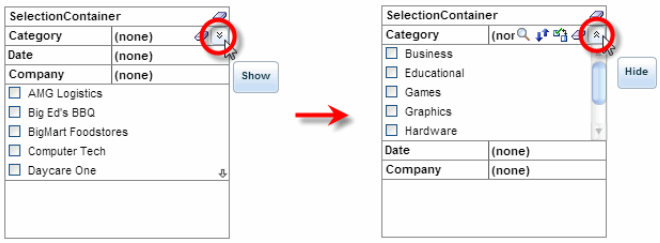
The following sections explain how to use a Selection Container.

See Also

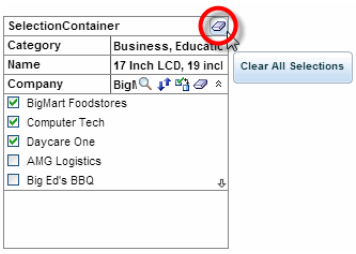
Selection Container, in *Advanced Dashboard Design*, for more information about Selection Containers.

Using Contained Selections

To display a selection component in a Selection Container, press the 'Show' button in the selection component title bar. This expands the selection component and its toolbar. (See *Selection Lists*, *Selection Trees*, and *Range Sliders* for information on how to use these selection components.) To collapse a selection component in a Selection Container, press the 'Hide' button.



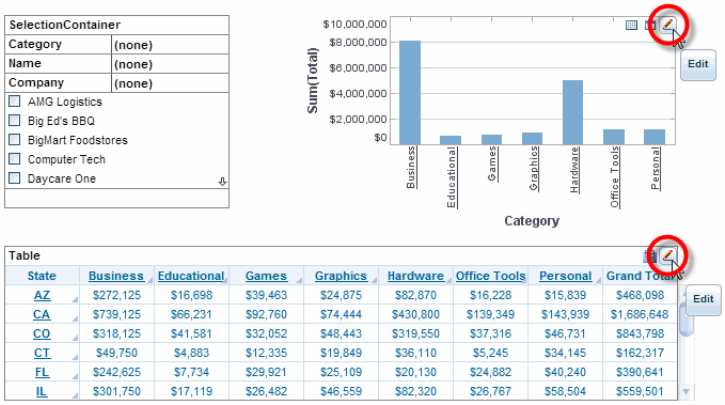
To clear the filters on all selection components in the Selection Container, press the 'Clear All Selections' button in the Selection Container title bar.



Adding Contained Selections

For certain dashboards, you may be able to add new selection components into an existing Selection Container. To do this, follow the steps below:

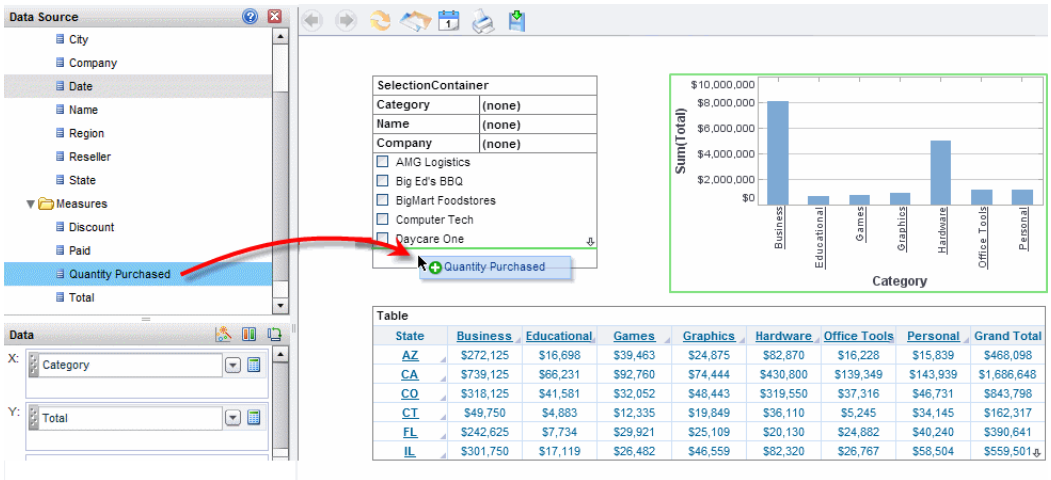
1. Press the 'Edit' button in a Chart or Crosstab component on the dashboard.



This opens the Chart Editor or the Crosstab Editor.

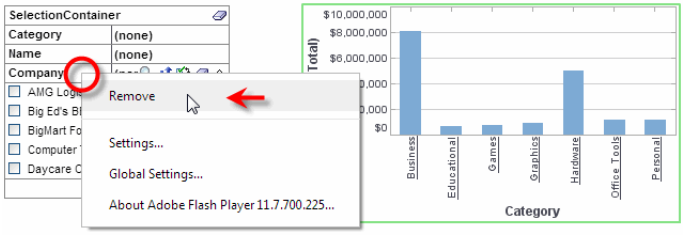
Note: If no 'Edit' button is available on any Chart or Crosstab, you cannot add a new selection into the Selection Container.

2. Drag the desired fields from the **Data Source** panel into the Selection Container.



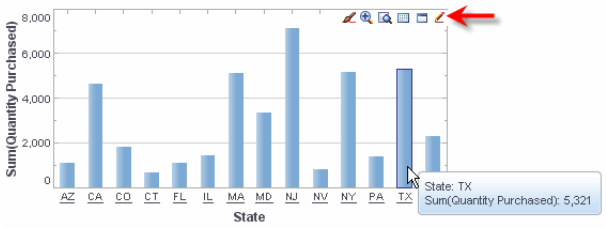
String-type fields are added to the Selection Container as Selection Lists, while numerical-type and date-type fields are added to the Selection Container as Range Sliders.

3. To remove a selection component from the Selection Container, right-click the component and select ‘Remove’.



Charts

Select one or more data groups in a Chart component to view the Chart toolbar. (You can select data groups by dragging across the groups on the chart, by clicking on group, or by Shift-clicking or Ctrl-clicking to select multiple groups.)



The following toolbar options may be available:

- ‘Edit’: Opens the Chart Editor to modify the chart. See *Charts* in the *Advanced Dashboard Design* for more information.
- ‘Max Mode View’: Displays the chart in a floating window that you can resize.
- ‘Chart Data’: Displays a tabular listing of the chart’s summarized data that you can export to Excel.
- ‘Show Details’: Displays the detail records underlying the chart’s summarized data that you can export to Excel.
- ‘Zoom’: Zooms the chart to display only the selected data groups. See *Zooming a Chart* for more details.
- ‘Exclude’: Zooms the chart to exclude the selected groups (i.e., displays only the unselected groups). See *Zooming a Chart* for more details.
- ‘Brush’: Highlights the selected data groups across all charts in which the data occurs. See *Brushing a Chart* for more information.

To save an individual Chart as an image, right-click the Chart and select 'Save Image As' from the context menu.

The following sections explain how to format chart labels and set other chart properties.

See Also

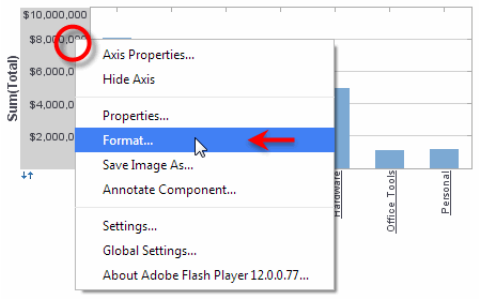
Changing the Chart View, in *Advanced Dashboard Design*, for more detailed information on these features.

Grouping Chart Labels into Named Groups, in *Advanced Dashboard Design*, for information on agglomerating chart categories.

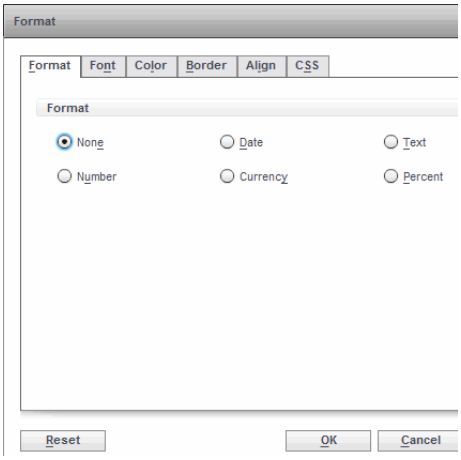
Formatting Chart Elements

Chart formatting includes data formats (date, number, currency, etc.), borders, colors, fonts, and other aspects pertaining to the visual presentation of the Chart.

To set formatting for a particular aspect of a Chart, right-click on the aspect of the chart that you want to format (axis title, axis labels, plot area, etc.), and select 'Format'.



This opens the 'Format' dialog box.

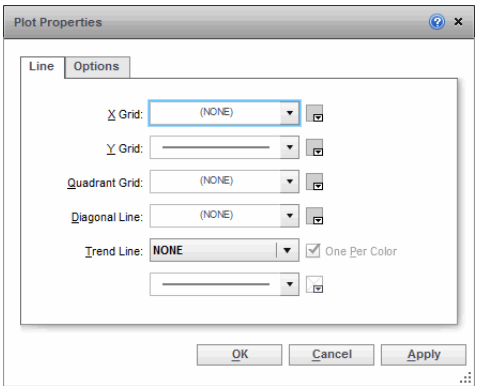


See *Format Dialog Box* in *Advanced Dashboard Design* for full information about the dialog box properties.

Setting Chart Plot Properties

Plot properties for a Chart include grid-lines, trend-lines, banding, transparency, and other features that pertain to the visual appearance of the plot region.

To set plot properties for a Chart, right-click on the chart and select 'Plot Properties'. This opens the 'Plot Properties' dialog box.



See *Editing Plot Properties* in *Advanced Dashboard Design* for full information about the dialog box properties.

Managing the Server

```
Only users who have administrator privileges can change server
settings.
```

There are several key aspects to managing the Style Intelligence server and Style Scope server: Configuring security, scheduling tasks and cycles, and deploying assets. These topics are discussed in the following sections.

Configuring Security

There are three steps to setting up security:

1. *Specifying a Security Provider*
2. *Defining Users, Roles, and Groups*
3. *Setting Permissions*

The following sections discuss these steps in order.

Specifying a Security Provider

A security provider is a module that handles security operations for the Style Intelligence or Style Scope server. You can specify independent security providers to provide the following functions:

- **Authentication**: Retrieving users, roles, and groups; authenticating logins.
- **Authorization**: Storing and retrieving permissions for different components and functions.

To set security provider options, follow the steps below:

See *Starting the Server* for information on how to start the server.

1. Open Enterprise Manager. (If the server is running, simply enter the appropriate URL in any browser. The default URL is `http://localhost:8080/sree/EnterpriseManager`).

2. Select the 'Server' > 'Security' option under the **Server** tab.



3. Select 'Enable Security' to activate security settings.

4. In the 'Authentication' menu, select the desired method for verifying user login credentials:

   a. If you choose the 'Default' method for authentication, proceed to define your users, roles, and groups in *Defining Users, Roles, and Groups*.

b. If you choose the ‘LDAP’ option for authentication, specify the connection information for the LDAP server containing your user, role, and group information. (See *Configuring an LDAP Security Provider* in *Advanced Server Administration* for more details on the settings.) Then proceed to assign permissions in *Setting Permissions*.

5. In the ‘Authorization’ menu, choose between ‘Default’ and ‘LDAP’. The ‘Default’ option for authorization is **strongly recommended**, as this avoids making modifications to your LDAP schema.

See Also
Security, in *Advanced Server Administration*, for more details.

Defining Users, Roles, and Groups

If you are using the ‘Default’ security provider for authentication (see *Specifying a Security Provider*), create users, groups, and roles under Enterprise Manager’s **Users** tab. The following sections explain how to do this.

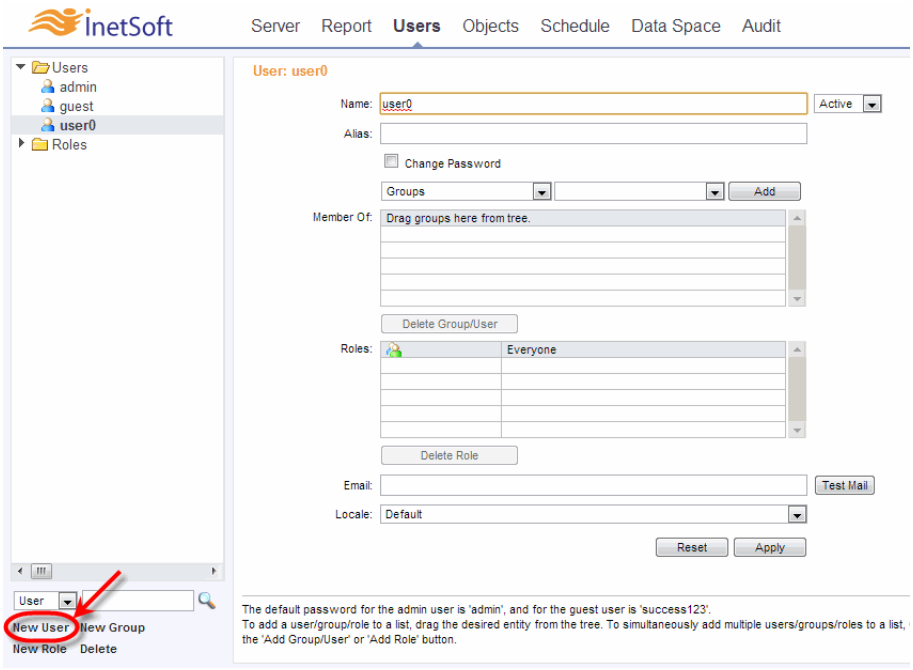
- *Creating a User*
- *Creating a Group*
- *Creating a Role*

Creating a User

There are two embedded users, ‘admin’ and ‘guest’; ‘admin’ is a super-user with the role of ‘Administrator’ and ‘guest’ is a user with the role of ‘Everyone’. The default passwords are ‘admin’ and ‘success123’, respectively.

To create a new user, follow the steps below. (You can create and edit users in Enterprise Manager only when using the ‘Default’ authentication provider.)

1. From Enterprise Manager’s **Users** tab, press the ‘New User’ link below the navigation tree. This creates a user with the default name ‘userN’ (user0, user1, etc.).



2. Type in the ‘Name’ field to change the name of the user.

3. Select the ‘Change Password’ option to change the user’s password.

4. Select ‘Active’ or ‘Inactive’ from the right-side menu. A user marked as ‘Inactive’ is not able to log into the Portal.

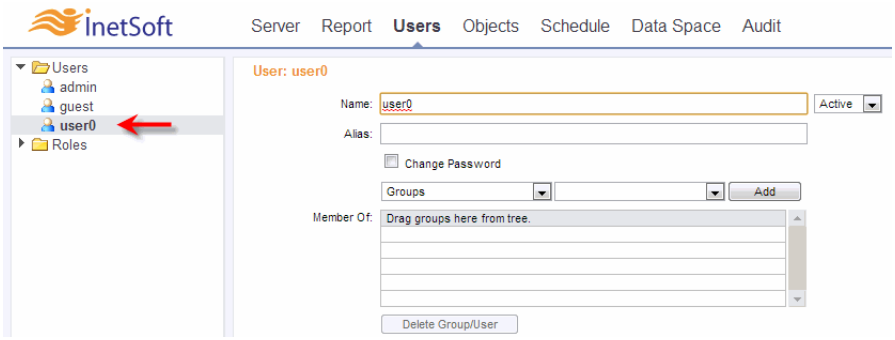
5. Drag groups from the left-side tree into the ‘Member Of’ box to assign the user to the designated groups. (Alternatively, select ‘Groups’ from the menu above the ‘Member Of’ box, choose the desired group, and press ‘Add’.)

6. Drag roles from the left-side tree into the ‘Roles’ box to assign roles to the user. (Alternatively, select ‘Roles’ from the menu above the ‘Member Of’ box, choose the desired role, and press ‘Add’.)

7. Specify an email address for the user in the ‘Email’ field.

8. Press ‘Apply’ to save your changes.

To edit a user, expand the ‘Users’ node on the tree and click on the user name. Make any desired changes, and press ‘Apply’. (A user may be listed directly under the ‘Users’ node or be listed within a group under the ‘Users’ node.)

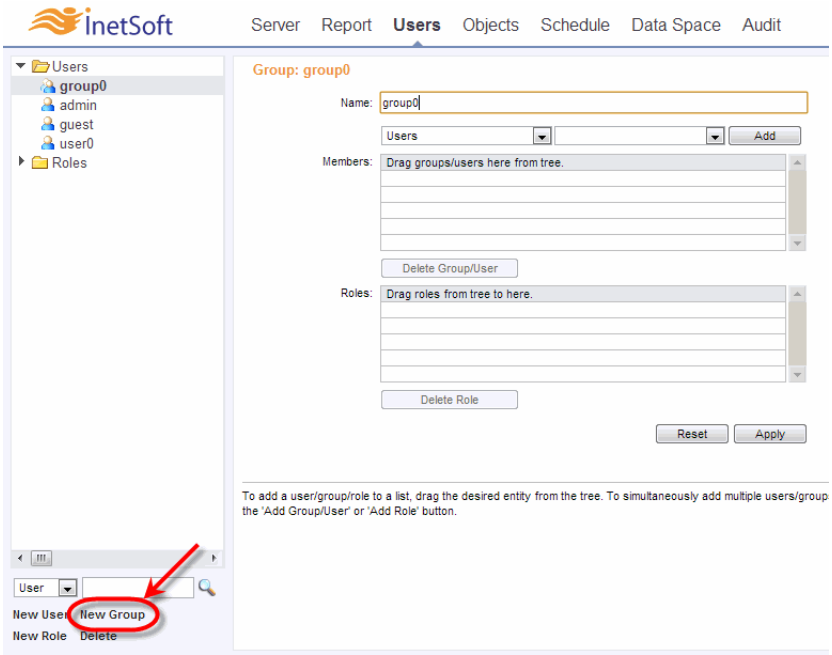


Creating a Group

A group is a collection of users, such as a project team, a department, or a geographical location. (You can create and edit groups in Enterprise Manager only when using the 'Default' authentication provider.)

To create a new group, follow the steps below:

1. From Enterprise Manager's **Users** tab, press the 'New Group' link below the navigation tree. This creates a group with the default name 'groupN' (group0, group1, etc.).



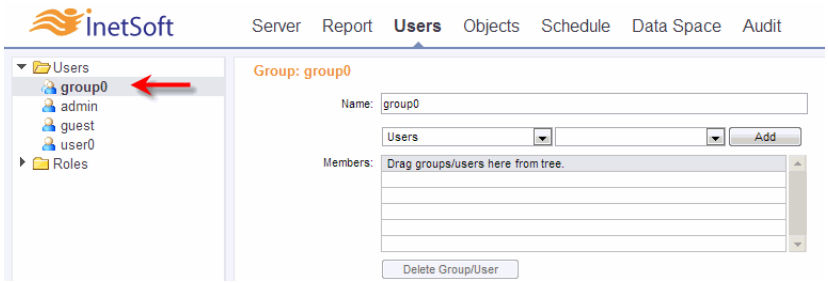
2. Type in the 'Name' field to change the name of the group.

3. Drag users and/or groups into the 'Members' box. This adds the individual users and users within the groups to the new group. (A user/group can be added to multiple groups.)

4. Drag roles into the ‘Roles’ box. All of the users in this group will be assigned the specified roles.

5. Press ‘Apply’ to save your changes.

To edit a group, expand the ‘Users’ node on the tree, and select the desired group. Make the desired changes and press ‘Apply’.



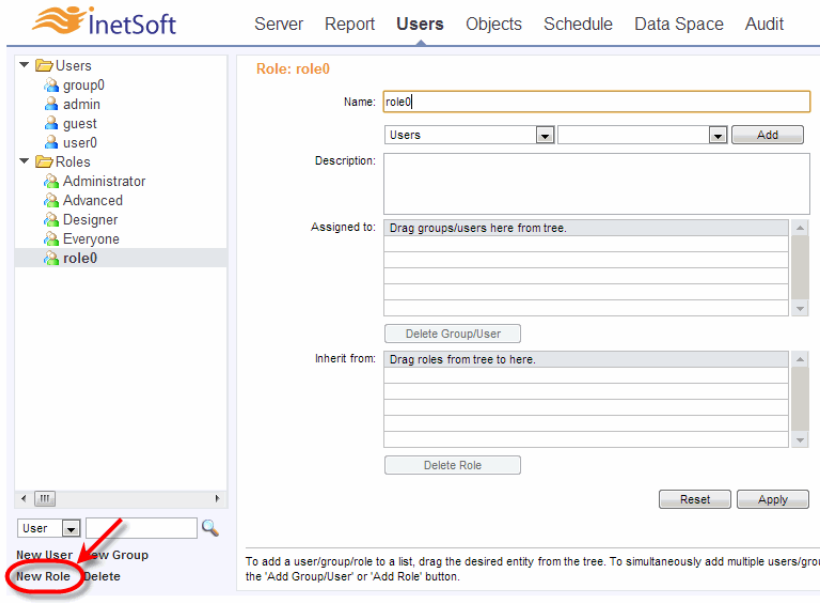
Creating a Role

There are two embedded roles, ‘Everyone’ and ‘Administrator’. Every user is by default assigned to the ‘Everyone’ role; it is typically used to expose a resource to all users. The ‘Administrator’ role is a super-user who has access to all resources. Any user with the ‘Administrator’ role can log into Enterprise Manager.

A role (e.g., Manager, Developer, Analyst, etc.) can be used to assign permissions to a set of users who perform a similar set of functions. You can create and edit roles in Enterprise Manager only when using the ‘Default’ authentication provider.

To add a new role, follow the steps below:

1. From Enterprise Manager’s **Users** tab, press the ‘New Role’ link below the navigation tree. This creates a role with a default name ‘roleN’ (role0, role1, etc.).



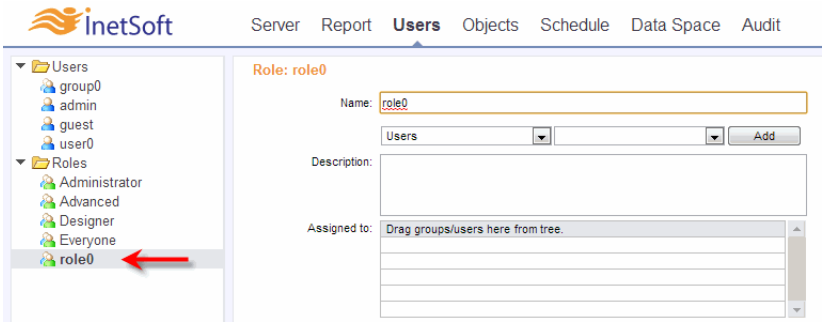
2. Type in the ‘Name’ field to change the name of the role.

3. Drag users and/or groups into the ‘Assigned To’ box. This assigns all of the individual users and users within the group to this role.

4. Drag roles into the ‘Inherit From’ box. The new role will inherit all permissions from the roles specified in this list.

5. Press ‘Apply’ to save your changes.

To edit a role, expand the ‘Roles’ node on the tree, and select the desired role. Make any desired edits and press ‘Apply’.



Setting Permissions

To set permissions for individual dashboards, reports, or repository folders, use the Enterprise Manager **Report** tab. See *Setting Repository Permissions*. To set all other permissions (e.g., access to data models, queries, Data Worksheets, etc.), use the Enterprise Manager **Objects** tab. See *Setting Object Permissions*.

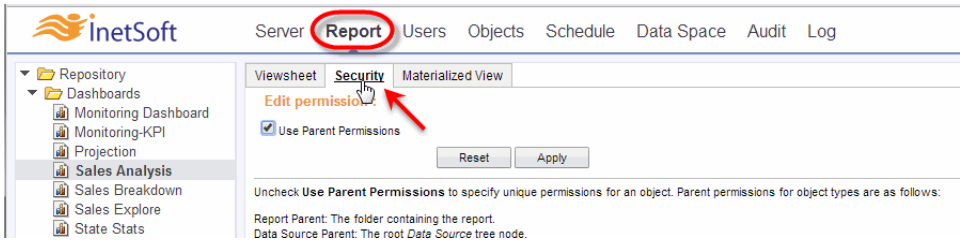
Setting Repository Permissions

Repository folders are *logical* groupings, and do not represent file system directories.

To set permissions for dashboards, reports, and repository folders, select the **Report** tab in Enterprise Manager and expand the ‘Repository’ node on the tree. The repository is a collection of dashboards and reports organized into one or more folders.

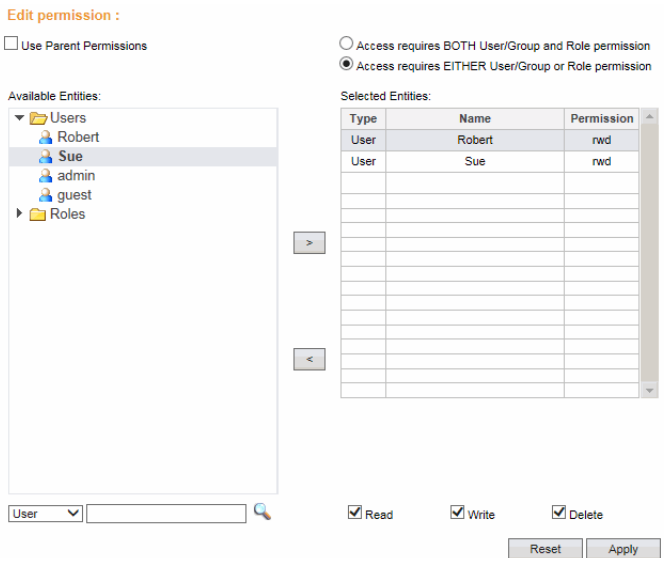
To set the permissions for an individual dashboard, report, or folder, follow the steps below:

1. Select the dashboard, report, or folder from the tree, and press the **Security** tab.



2. Deselect the 'Use Parent Permissions' check-box.

3. Drag the desired users/roles/groups from the 'Available Entities' list to the 'Selected Entities' list.



4. Select 'Access requires both User and Role permission' to specify that access to the dashboard, report, or folder is granted only if the current user has **both** user **and** role permissions; i.e., both the user's name (or group) and at least one of the user's roles appear in the 'Selected Entities' table.

   Select 'Access requires either User or Role permission' to specify that access to the dashboard, report, or folder is granted if the current user has **either** user **or** role permissions; i.e., either the user's name (or group) or at least one of the user's roles appear in the 'Selected Entities' table.

5. Select the appropriate permission detail:

```
r = read
w = write
d = delete
```

6. Press 'Apply' to save your changes.

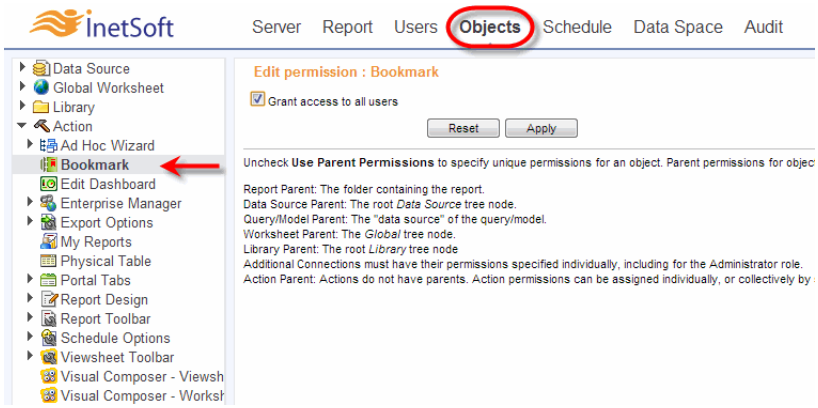
See Also

Repository Permissions, in *Advanced Server Administration*, for further details.

Setting Object Permissions

The procedure for setting object permissions is the same for all components. For example, to set permissions for the 'Bookmark' feature, follow the steps below:

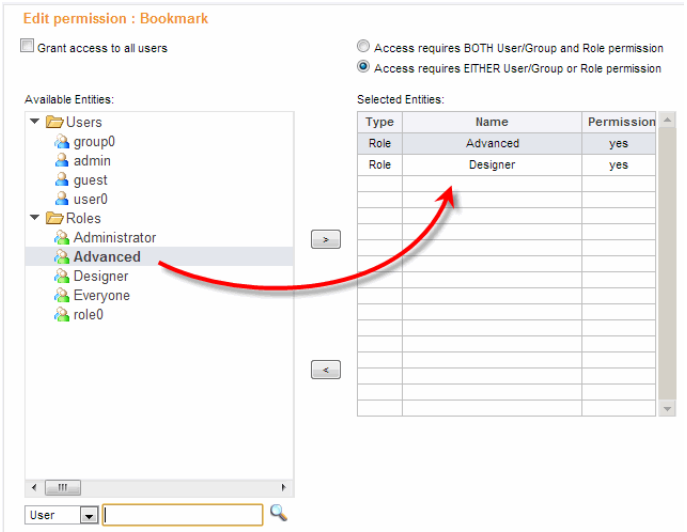
1. Select the **Objects** tab in Enterprise Manager, and select the 'Bookmark' item in the tree.



2. If the 'Grant access to all users' check-box is checked, click the box to uncheck it.

   Note: Depending on the particular component, the check-box may be labeled 'Grant access to all users', 'Deny access to all users', or 'Use parent permissions'. In all cases, you must uncheck the box in order to specify individual permissions. See *Parent Permissions* in *Advanced Server Administration* for more information.

3. Drag the desired users/roles/groups from the 'Available Entities' list into the 'Selected Entities' list.



4. Select ‘Access requires both User and Role permission’ to specify that access to the object or component is granted only if the current user has **both** user **and** role permissions; i.e., both the user’s name (or group) and at least one of the user’s roles appear in the ‘Selected Entities’ table.

   Select ‘Access requires either User or Role permission’ to specify that access to the object or component is granted if the current user has **either** user **or** role permissions; i.e., either the user’s name (or group) or at least one of the user’s roles appear in the ‘Selected Entities’ table.

5. Select the appropriate permission detail:

   ```
   r = read
   w = write
   d = delete
   ```

   The write and delete options are unavailable for some components.

6. Press ‘Apply’ to save your changes.

Scheduling

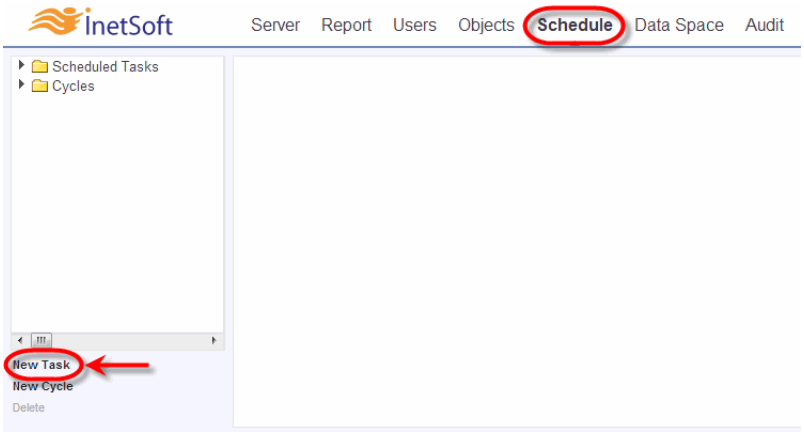
Style Intelligence and Style Scope include a Scheduler application that allows you to automatically generate and email dashboards, and to define cycles for materialized views. The following sections explain how to use the Scheduler.

Creating a Scheduled Task

Users can also create scheduled tasks in the Portal. See *Scheduling Dashboards and Reports*.

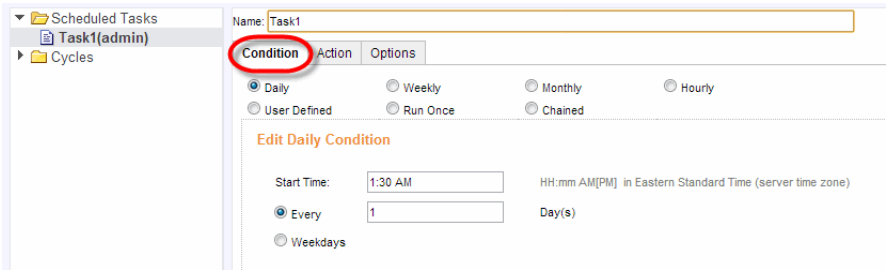
To create a scheduled task, follow the steps below:

1. Select the **Schedule** tab in Enterprise Manager and press the 'New Task' link below the navigation tree. This creates a new task, with the default name 'Task n' (e.g., Task1, Task2, Task3, etc.).

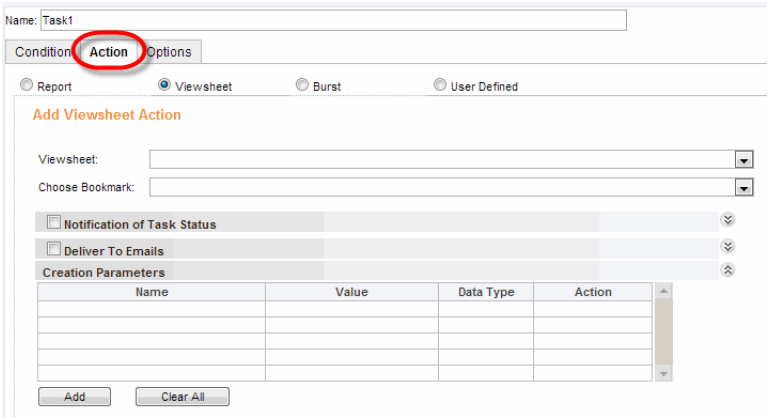


2. Enter a name for the scheduled task in the 'Name' field.

3. Select the **Condition** tab and specify a condition to determine when the task will execute. See *Scheduler Conditions* in *Advanced Server Administration* for details.



4. Select the **Action** tab and specify the type of action to take when the scheduled task executes. See *Scheduler Actions* in *Advanced Server Administration* for details.

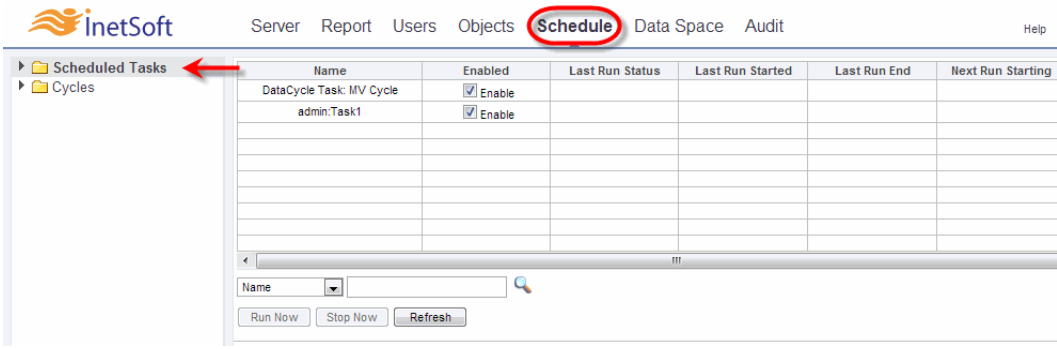


5. Press the 'Save' button at the bottom to save the task.

Monitoring a Scheduled Task

To monitor the status of scheduled tasks, follow the steps below:

1. Select the **Schedule** tab in Enterprise Manager and click on the 'Scheduled Task' folder.



   This displays a table of all scheduled tasks, including the start time, end time, status of last run, and start time for the next run.

2. To temporarily disable or enable a task, click the appropriate checkbox in the 'Enabled' column.

3. To execute a task on-demand, click the task name to select it and press the 'Run Now' button.

4. To halt a task in progress, press the 'Stop Now' button.

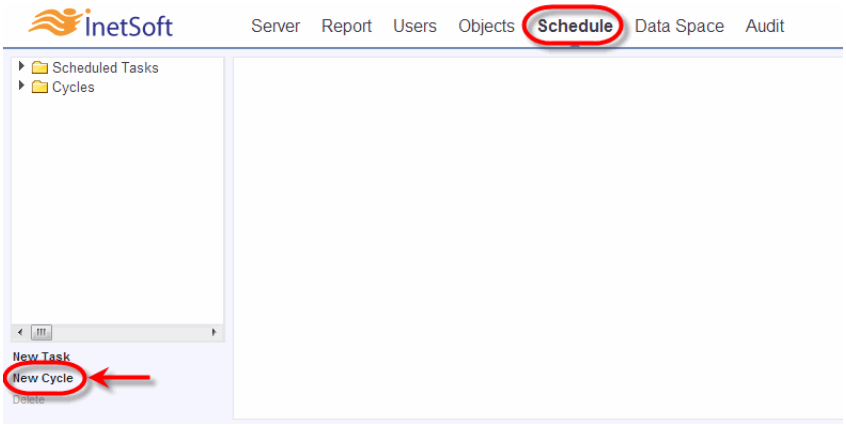
Creating a Scheduled Cycle

A Scheduler cycle is a time condition that determines when materialized view generation or report pre-generation should take place. (See

Materialized Views and *Report Pregeneration* in *Advanced Server Administration* for more information.)

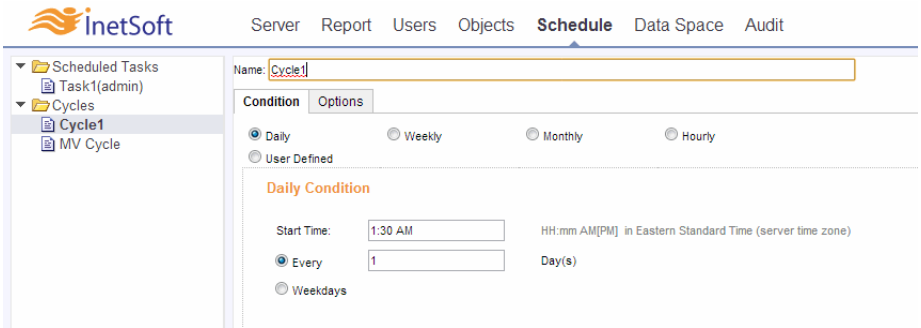
To create a new cycle, follow the steps below:

1. Select the **Schedule** tab in Enterprise Manager and press the 'New Cycle' link below the navigation tree. This creates a new cycle, with the default name 'Cycle n' (e.g., Cycle1, Cycle2, Cycle3, etc.).



2. Enter a name for the new cycle in the 'Name' field.

3. Select the **Condition** tab and specify a condition to determine when the cycle will execute. See *Scheduler Conditions* in *Advanced Server Administration* for more details.



4. Press the 'Save' button at the bottom to save the cycle.

Incrementally Deploying Assets

An **asset** is any dashboard, report, or component (table style, query, Data Worksheet, etc.). To move assets from one server environment to another server environment (e.g., from a test server to a production server), use the incremental deployment features described in the following sections.

Exporting Assets

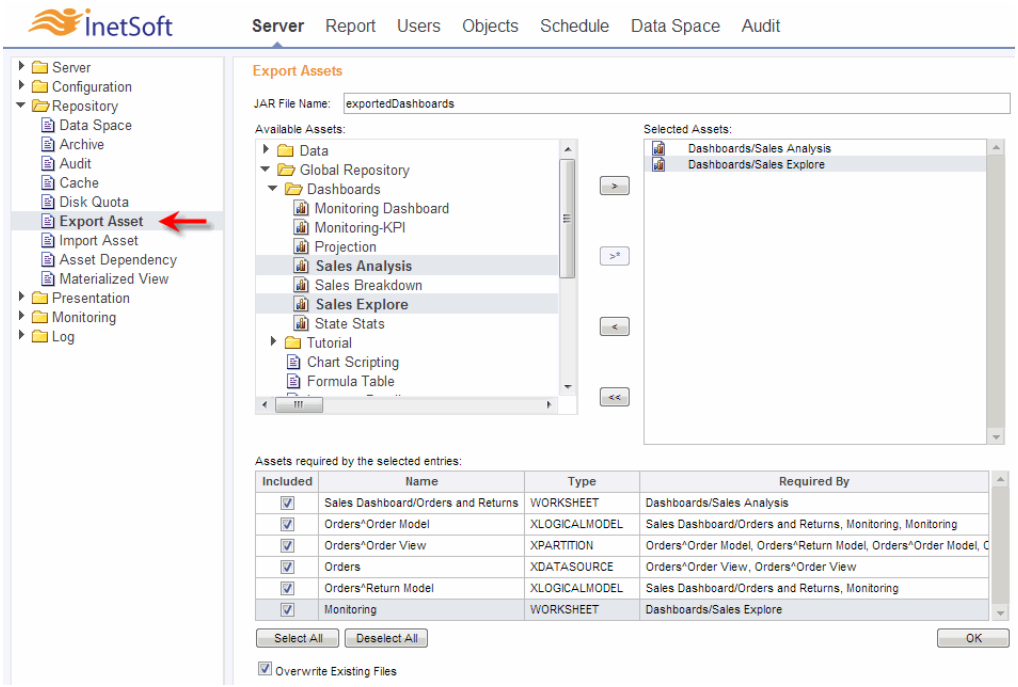
To export a set of assets into an archive (JAR) file, follow the steps below:

1. In Enterprise Manager, open the 'Repository' > 'Export Asset' page under the **Server** tab.

2. Enter a name for the exported JAR file in the 'JAR File name' field.

3. Select one or more assets in the 'Available Assets' panel.

4. Press the arrow button to move the selected assets to the 'Selected Assets' panel.



The 'Assets required by the selected entries' table at the bottom of the page updates to list the assets used by the entries in the 'Selected Assets' panel.

5. Mark the appropriate checkboxes in the 'Included' column of the 'Assets required by the selected entries' table to include the corresponding assets in the archive file.

6. Check 'Overwrite existing files' if the assets in the archive should *by default* overwrite existing assets of the same name when imported into the new environment. This behavior can be modified if needed when the assets are imported.

7. Press 'OK' to save the JAR file. Depending on your browser's configuration, you will either be prompted to save the archive file in the browser's designated download directory, or you will be prompted to choose a location in which to save the export file.

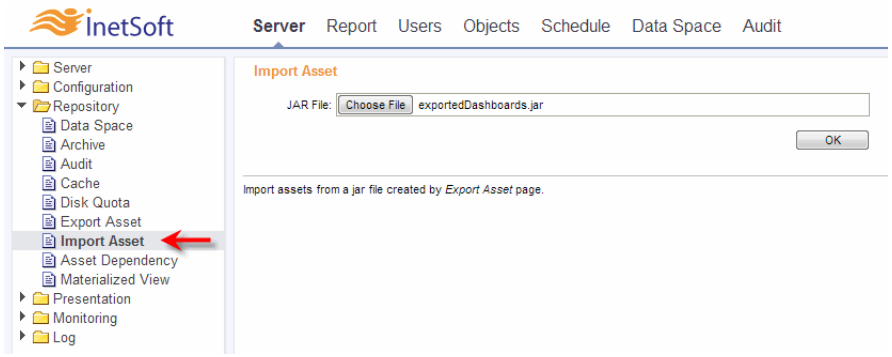
See *Importing Assets* for instructions on how to import the assets from the JAR file into another server environment.

Importing Assets

To import a set of assets from an archive file generated by the 'Export Asset' feature (see *Exporting Assets*), follow the steps below:

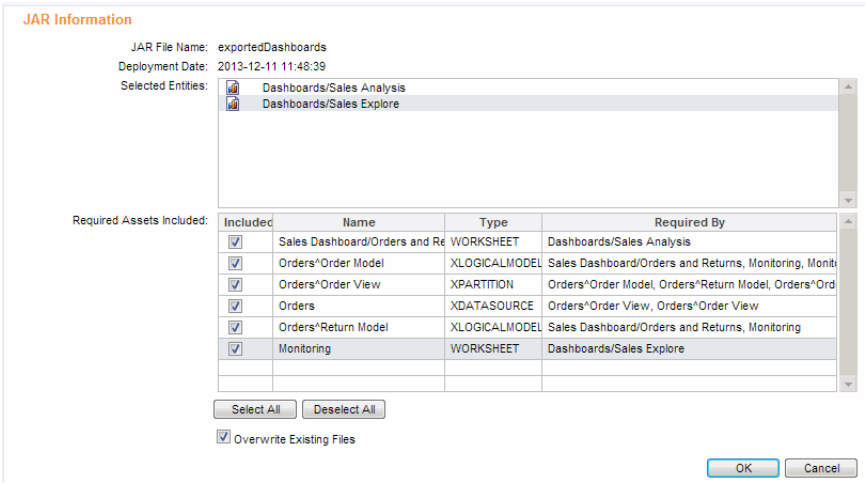
1. In Enterprise Manager, open the 'Repository' > 'Import Asset' page under the **Server** tab.

2. On the 'Import Asset' page, press 'Choose File' and select the JAR file containing assets to import.



3. Press 'OK'.

4. This opens the 'JAR Information' page displays the asset contents of the specified JAR file.



The items listed in the ‘Selected Entities’ panel will all be imported. The items listed in the ‘Included Dependent Assets’ table are required by items in the ‘Selected Entities’ panel, and will be imported by default. However, if one or more of these items already exist in the current server environment, you can choose not to import those items.

5. Click the ‘Include’ box next to assets in the ‘Included Dependent Assets’ table that you wish to import.

6. Select ‘Overwrite existing files’ if the assets in the archive should overwrite existing assets of the same name upon import. The default setting of the ‘Overwrite existing files’ checkbox is specified by the export settings under which the archive was saved. (See *Exporting Assets*.)

7. Press ‘OK’ to import these assets into the current environment.

Advanced Data Modeling

Style Studio contains data modeling tools that enable you to rapidly connect to data sources, graphically construct queries and data models, and implement dynamic security and filtering. With Style Studio, you do not need to be a business intelligence specialist to become an efficient data modeler.

1 Introduction

The data modeling features in Style Studio allow you to connect to various data sources, and create queries, data models, and virtual private models (VPMs). These assets can be used directly to supply data to reports and dashboards. They also lay the foundation for Data Block™ technology, which allows you to create sophisticated data mash-ups from heterogeneous data sources. (For more information on Data Worksheets, please see the *Advanced Data Mashup*.)

Style Studio's data modeling features give you ready access to data stored in relational databases, objects, and flat files. Databases include data warehouses, data marts, mainframes, operational data stores (ODS), multi-dimensional databases (OLAP), and transactional databases (OLTP). Objects include web services, XML, and plain old java objects (POJO). Flat files include spreadsheets, CSV, and text.

2 Data Modeling Features

This chapter provides an overview of the data modeling features offered by Style Studio.

2.1 Accessing a Data Source, Query, Model, VPM

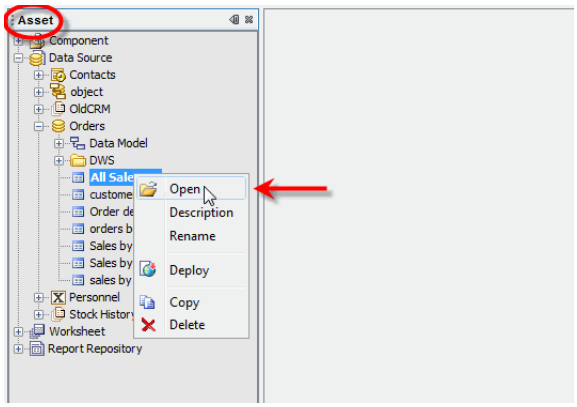
To access an existing data source, query, model, or VPM, follow the steps below:

1. From the Style Studio *Window* menu, select 'Asset Tree'. This opens the **Asset** panel.

2. In the **Asset** panel, expand the 'Data Source' node and open the desired asset for editing. Follow the steps below:

    a. To open a data source for editing, double-click the desired data source node, or right-click and select 'Open' from the context menu.

    b. To open a query for editing, first expand the parent data source. Then double-click the desired query node, or right-click and select 'Open' from the context menu.



    c. To open a data model for editing, first expand the parent data source and expand the 'Data Model' node. Then double-click the desired physical view or logical model node, or right-click and select 'Open' from the context menu.

    d. To open a VPM for editing, first expand the parent data source. Then double-click the desired VPM node, or right-click and select 'Open' from the context menu.

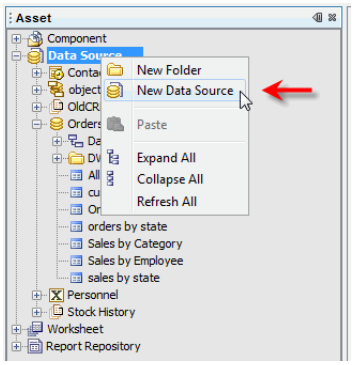
To create a new data source, query, model, or VPM, follow the steps below:

1. Click the ‘New’ button in the Style Studio toolbar. This opens the ‘New Asset’ dialog box.

2. Expand the ‘Data Source’ node, and select the component you wish to create.

3. Make any required selections in the right ‘Type’ panel, and click ‘OK’.

Alternatively, you can right-click a node on the **Asset** tree, and select ‘New Data Source’, ‘New Query’, ‘New Folder’, etc.



See the related links for further information about creating different assets.

See Also

Creating a New JDBC Data Source, for information on connecting to a database.

Independent Query, for information on creating a query with the Query Wizard.

Data Models, for information on creating a data model.

Virtual Private Model Security, for information on creating a data-level filter.

2.2 Configuring Data Source and Query Registries

Style Studio uses two XML repository files to store the data source/model and query definitions (*datasource.xml* and *query.xml*, respectively). The first time you launch Style Studio, a dialog box prompts you to specify the locations of these files. You can change the repository directory location at a later time by selecting the *File* > *Configure* option from the Style Studio menu bar. See *Configuring Style Studio* for more information.

2.3 The Data Modeling Toolbar

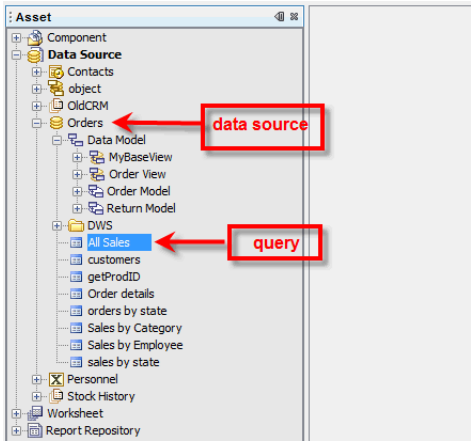
The data modeling options on the toolbar may change slightly depending on the data source type.

Table 1. Data Modeling Toolbar Buttons

	Save the selected data source or query.
	Specify options for the selected query. (Max Rows, Timeout)
	Preview the selected query.
	Test the connection to the selected data source.
	Edit the parameter definitions for the selected query.
	Add a new physical view.
	Add a new logical view.
	Add a new Virtual Private Model.
	Refresh Style Studio with the current structure and meta-data of all the data sources.
	Populate the result column list for a stored procedure.
	Set Request Parameters for a query (SOAP, XML).

2.4 Data Source/Query Tree

The 'Data Source' node in the **Asset** panel lists the available data sources, together with their associated queries, models, and VPMs.



A *data source* provides connection information for a database. The information needed varies depending on the type of the database (e.g., a relational database requires a driver and URL). For more complex data sources, the data source objects also contain the meta-data information. The following types of datasources are supported.

	*Relational Databases* (JDBC: Oracle, DB2, MySql, SQL Server, Sybase, Access, etc.)
	*XML Data Source*

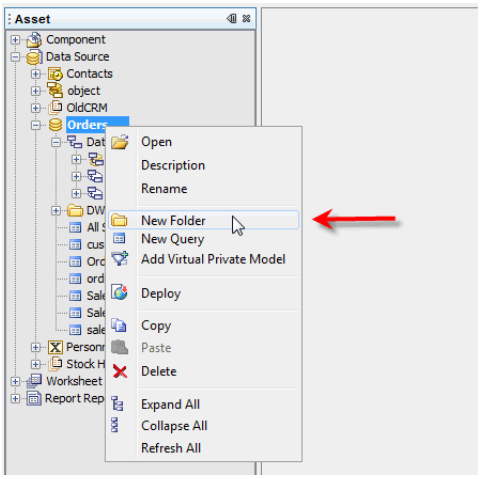
	*Generic Multidimensional Data Source* (XMLA)
	*Text Data Source* (CSV, Tab-delimited)
	*Web Service Data Source* (SOAP)
	*Java Object Data Source*
	*SAP Data Source*

Relational (JDBC) data sources may also have associated data models (discussed in the following sections). A data source can have multiple data models, but each data model can only use one data source. Likewise, each query is associated with one data source.

2.4.1 Grouping Queries Using Folders

You can organize queries for a given data source into folders. To create a new folder, follow these steps:

1. Right-click on the data source in the **Asset** panel, and select 'New Folder' from the context menu.



2. When prompted, enter a name for the new folder, and click 'OK'.

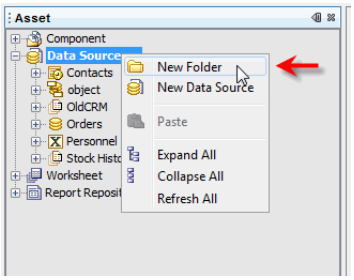
3. Drag the desired queries into the new folder.

2.4.2 Grouping Data Sources Using Folders

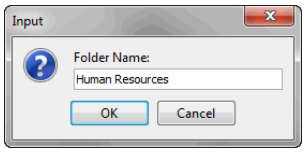
You can organize data sources into folders. This allows you to assign permissions to data sources on a folder-by-folder basis. (See *Component/ Object Permissions* in *Advanced Server Administration* for details on assigning permissions.)

To create a new data source folder, follow these steps:

1. Right-click on the 'Data Source' node in the **Asset** tree or right-click on an existing data source folder, and select 'New Folder' from the context menu.

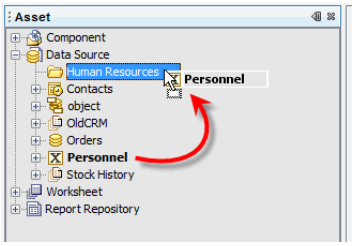


2. When prompted, enter a name for the new folder, and press 'OK'.



This creates a new folder on the tree under the 'Data Source' node.

3. Drag the desired data sources from the tree into the new folder.



Data sources within a given folder must have unique names.

To delete a data source folder, right-click on the folder and select 'Delete' from the context menu. This deletes the folder as well as all of the data sources contained within the folder.

See Also

Adding Additional Connections, for information about making multiple data source connections for a multi-tenant environment.

2.4.3 Deploying a Data Source, Query, or Data Model

You can easily deploy a data source, query, or data model to a repository other than the working repository (i.e., a *remote* repository) or to a JAR file for later use. See *Deploying an Asset* for more information.

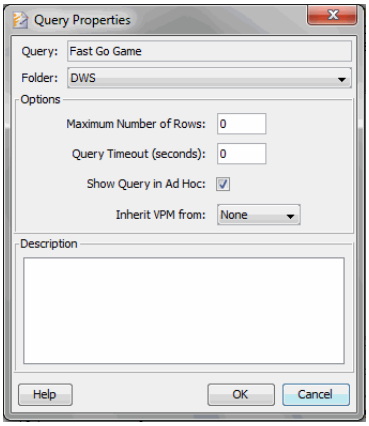
2.5 Advanced Toolbar Buttons

When you open a query for editing, the toolbar provides several query-specific buttons.

2.5.1 Query Properties



The ‘Query Properties’ button opens the ‘Query Properties’ dialog box. This allows you to place limits on queries that return a large result set by constraining the maximum number of returned rows and maximum execution time.



The following options are available:

- **Maximum Number of Rows**: The maximum number of records that the query should be permitted to request.
- **Query Timeout**: The maximum length of time for which the query should be permitted to execute. (After this period elapses, attempts will be made to terminate the query.)
- **Show Query in Ad Hoc**: Specifies that the query will be available to reports designed using the end-user Ad Hoc tool.
- **Inherit VPM from**: Specifies an existing physical view whose VPM filter will be applied to the query.

2.5.2 Query Parameters



The ‘Parameters’ button opens the ‘Query Variable Definition’ dialog box. This allows you to edit query parameters that have been used in the query condition. (See *Creating a New Query* for information on parameterizing a query.)

There are two options: ‘User variable’ and ‘Query variable’. The following sections explain how to use these parameter types.

Acquiring a Query Parameter from User Input

By default, when you add a parameter in a query condition, the user is prompted to enter the value at runtime. (To use another field of the query to provide an aggregate value for the right side of the condition, see *Acquiring a Query Parameter from a Query Field*.)

To customize user prompting when setting the parameter value by user input, follow the steps below:

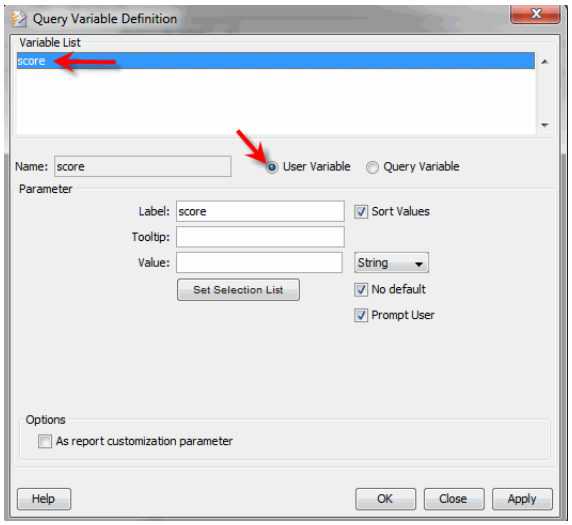
1. Open the desired query for editing. (Double-click the query in the **Asset** panel.)



2. Click the 'Parameters' button in the toolbar. This opens the 'Query Variable Definition' dialog box. The 'Variable List' panel at the top displays any query parameters that have been specified in the query condition.

3. Select the desired variable in the 'Variable List' panel.

4. Click the 'User Variable' button (if not already selected).



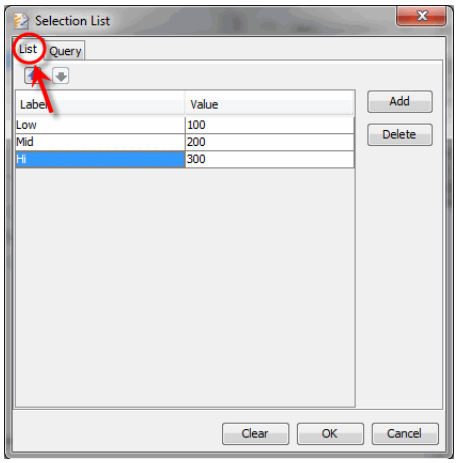
5. Click 'Set Selection List' to open the 'Selection List' dialog box. (The button's label is displayed in green if the selected parameter already has a list associated.)

6. To enter a fixed set of labels/values for the parameter, follow the steps below:

   a. Click the **List** tab.

   b. Click 'Add' to add a new label/value pair.

c. Enter the desired ‘Label’ and ‘Value’. (The ‘Label’ text is displayed to the user in the interface element; the corresponding ‘Value’ is assigned to the parameter.)



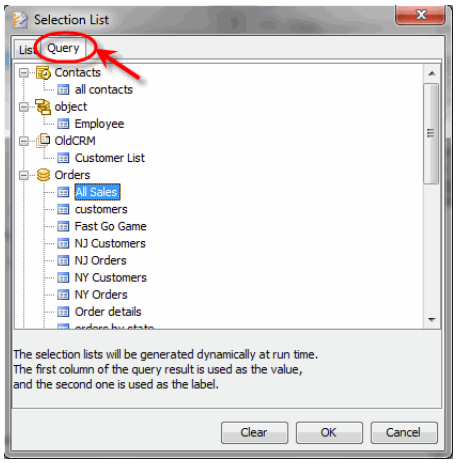
7. To acquire parameter labels/values from a different query at runtime, follow these steps:

Note: The query is used to prompt the user for input. If you want a query to automatically set the parameter value (without user input) see *Acquiring a Query Parameter from a Query Field*.

a. Click the **Query** tab.

b. Select the query that contains the label/value pairs.

The data in the first column of the query are used as the parameter values; the data in the second column of the query are used as the labels. (Query conditions can reference other parameters if those parameters possess default values.)



8. (Optional) Select ‘Allow multiple selections’ to permit the user to select multiple values for the parameter.

To remove values associated with a parameter, click the ‘Clear’ button in the ‘Selection List’ dialog box.

9. Click ‘OK’ to close the ‘Selection List’ dialog box.

10. Make the following optional selections in the ‘Query Variable Definition’ dialog box, if desired:



    a. Select ‘As customization parameter’ to expose the parameter as a customization parameter. Customization parameters are displayed when the user clicks the ‘Customize’ button on the report toolbar.

    b. Select ‘Prompt User’ if the user should be automatically prompted to enter a value for this parameter. (Disable this option for parameters set by script and drill-down reports.)

    c. Select ‘Sort Values’ if you want the list to be sorted alphabetically.

    d. Enter text in the ‘Tooltip’ field to display when the user hovers the mouse over the corresponding parameter control on the input form.

11. Click ‘OK’ to close the ‘Query Variable Definition’ dialog box.

Acquiring a Query Parameter from a Query Field

Instead of acquiring a query parameter value from the user, you can provide the value by aggregating a field of another query. To do this, follow the steps below:

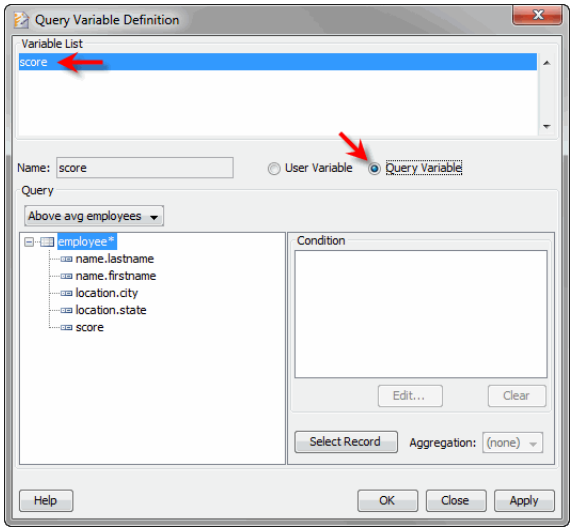
1. Open the desired query for editing. (Double-click the query in the **Asset** panel.)



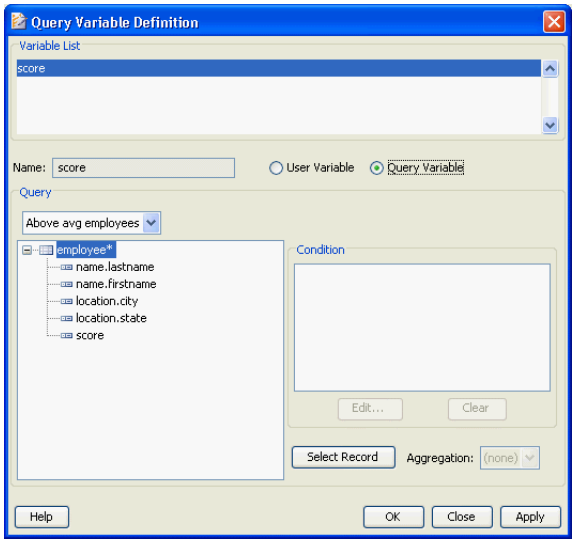
2. Click the ‘Parameters’ button in the toolbar. This opens the ‘Query Variable Definition’ dialog box. The ‘Variable List’ panel at the top displays any query parameters that have been specified in the query condition.

3. Select the desired variable in the ‘Variable List’ panel.

4. Click the ‘Query Variable’ button (if not already selected).



5. In the ‘Query’ panel, select a query from the top menu. This displays a list of query fields in the panel below.



6. Select the desired query field, and click ‘Select Record’. This places a marker (diamond) next to the selected field.

7. From the ‘Aggregation’ menu, choose a summarization method. If you select ‘None’, the first value from the query result set will be used.

8. (Optional) To filter the selected query, click on the query name, and then click the ‘Edit’ button below the ‘Condition’ panel. This opens the ‘Conditions’ dialog box, where you can specify a filtering condition.

2.6 Auto Back-Up and Recovery

When you save, delete, or rename an asset, a backup of the previous version of the *datasource.xml* or *query.xml* file will be stored in the repository. The old version will be stored as

```
backup/datasource_{timestampmillisec}.bak
```

or

```
backup/query_{timestampmillisec}.bak
```

By default, a maximum of 10 backups are stored, but this can be changed by setting `datamodel.backfile.count` in the *.stylereport* properties file to the desired integer (use `0` to disable the backup mechanism).

3 Relational Databases

You can access a relational database both through *data models* and *queries*. Data models and queries can also be combined with one another using the Data Block™ architecture provided by Data Worksheets (see *Advanced Data Mashup*).

- Data models offer maximum flexibility for developer and end-user self-service, and are recommended in most situations. See *Data Models* for more information.
- Queries give you precise control over requests sent to the database, and can be used to provide a fixed result set to developers or end users See *Independent Query* for more information.

3.1 Data Source Configuration

The examples in this Guide use the sample 'Orders' JDBC database.

Before you can build data models or queries, you need to configure a data source. A *data source* contains the connection information that allows Style Studio can connect to a database. Style Studio supports all relational databases with available JDBC drivers 3.0 and above. This includes all major commercial and open source databases such as Oracle, SQL Server, DB2, Sybase, Informix, MySQL, PostgreSQL, MS Access, and more.

3.1.1 Creating a New JDBC Data Source

A data source defines the connection and login information for a database. To create a new data source, follow the steps below.

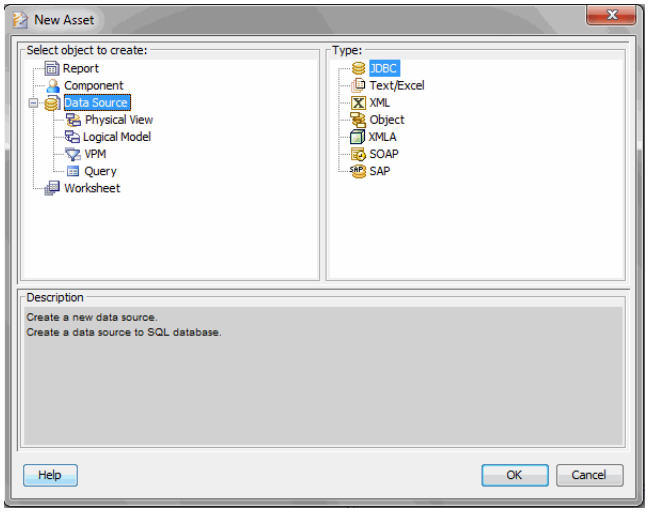


1. Click the 'New' button in the Style Studio toolbar. This opens the 'New Asset' dialog box.

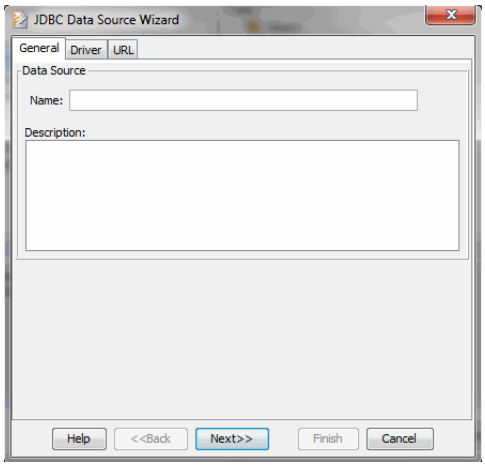


2. In the left panel of the 'New Asset' dialog box, click the 'Data Source' node on the tree.

3. In the right panel, select 'JDBC', and click 'OK'.



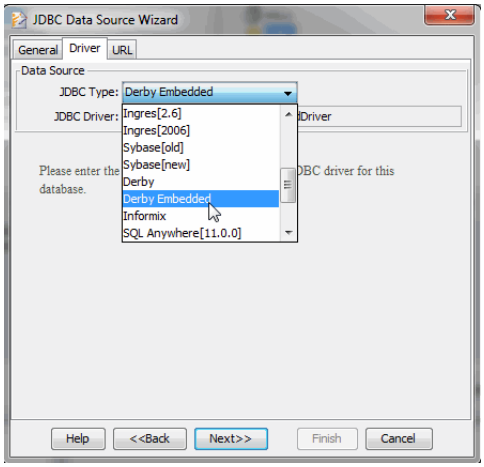
This opens the ‘JDBC Data Source Wizard’.



4. On the **General** tab, enter a name for the new data source. Click ‘Next’ to proceed.

Database vendors provide their own JDBC drivers. Make sure that the appropriate JAR files are on the CLASSPATH. See *Configuring the Style Studio Classpath* for more information. Style Studio will save the settings that you enter regardless of whether it locates the JDBC driver.

5. On the **Driver** tab, select the appropriate database name from the ‘JDBC Type’ menu. This automatically populates the ‘JDBC Driver’ field. Click ‘OK’ to proceed.



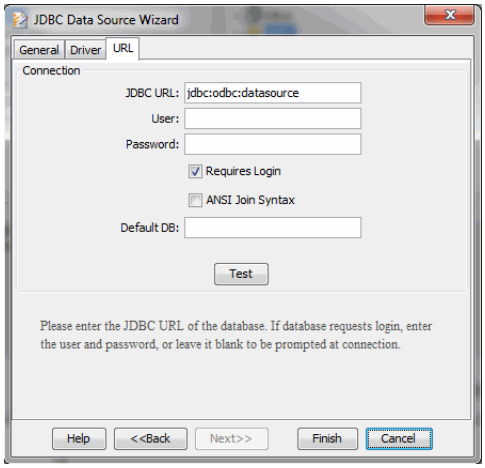
6. On the **URL** tab, enter the required connection information:

   a. The 'JDBC URL', 'User', and 'Password' information should be provided by your database administrator, if needed.

   b. (Optional) Select 'Requires Login' to specify that a user ID and password are required to connect to the database. For most databases, this option should be enabled.

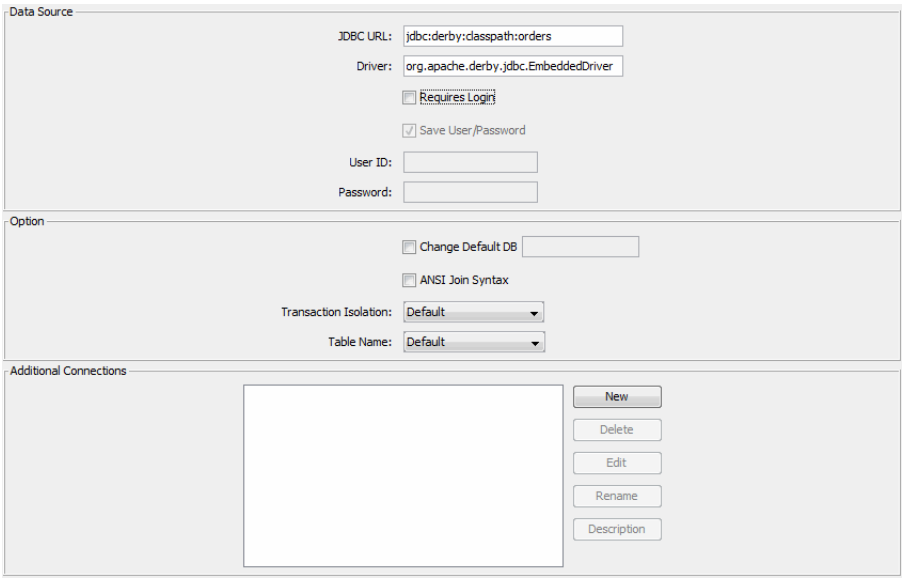
   c. (Optional) Select 'ANSI Join Syntax' to specify that queries should use the ANSI standard.

   d. (Optional) In the 'Default DB' field, enter the default schema name with which the login and password are associated.



7. Click 'Test' to verify that Style Studio can successfully connect to the database.

8. If the test was successful, click ‘Finish’. This displays the datasource definition page.



9. The ‘Transaction Isolation’ option sets the database transaction isolation level. For information about the ‘Additional Connections’ panel, see *Adding Additional Connections*.

10. The ‘Table Name’ option allows you to select how table names are represented in the SQL that is sent to the database.

    For example, if you select the ‘Schema.Table’ option, the SQL string will refer to tables by both the schema name and the table name. For a schema called “ORDERS”, table names would therefore appear in SQL as `ORDERS.CATEGORIES`, `ORDERS.PRODUCTS`, and so on. If you instead select the ‘Table’ option, the SQL string will refer to tables by just the table name, e.g., `CATEGORIES`, `PRODUCTS`, and so on.

11. Click the ‘Save’ button in the Style Studio toolbar to save the new data source to the *datasource.xml* file.

The new data source is now listed under the ‘Data Source’ node of the **Asset** panel.

Limitations of Hadoop Hive Data Sources

The following limitations exist when using the Hadoop Hive JDBC-type data source:

- Data models are not supported for Hadoop Hive data sources.

- Tables and columns in a Hadoop Hive database cannot be directly bound to tables and columns in reports or Data Worksheets. (You must create queries to retrieve the data.)
- Conditions in reports, Data Worksheets, and Viewsheets cannot be pushed to the Hadoop Hive database, and must therefore always be post-processed.
- The 'Browse Data' option is not available when designing conditions for a Hadoop Hive data source query.

Other Data Source Issues

In certain cases, divergence between the time zone for a MySQL database server and the Style Intelligence server may create a time-shift in the data retrieved. To correct this, explicitly specify the time zones for the Style Intelligence server and MySQL server using the `local.timezone` and `mysql.server.timezone` properties, respectively. See *Miscellaneous Properties* in *Advanced Server Administration* for details.

3.1.2 Parameterizing the Data Source

You can make the data source URL dynamic (in whole or part) by using parameters. To place a parameter in the URL, insert "`$(parameter name)`" at the desired location in the 'JDBC URL' string.

At runtime, Style Intelligence will attempt to acquire the value of this parameter by testing the following sequence of available sources:

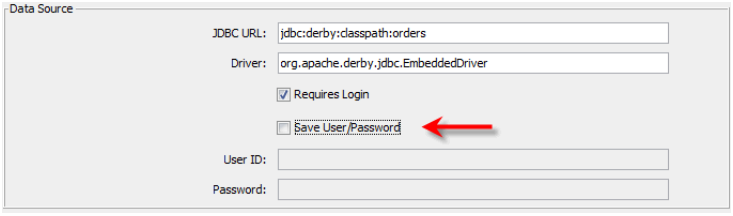
- Test for parameter stored as JVM system variable.
- Test for parameter stored as report parameter in the *sree.properties* file.
- Test for parameter passed within URL string.

If Style Intelligence cannot locate the parameter value in the above locations, it will by default prompt the user for the value.

3.1.3 Assigning User-Specific Data Source Login Credentials

You can allow different users to connect to the data source with user-specific login credentials. To configure user-specific login credentials, follow the steps below:

1. If a static username and password has been set for the data source, remove the static username and password.



2. Configure an SSO request filter to inject the following properties into the user Principal object:

   a. Username Parameter Name: `_Db_User_{Datasource Name}`

   b. Password Parameter Name: `_Db_Password_{Datasource Name}`

   See *Session-Based Single Sign-On* in *Advanced Server Integration* for information on how to create a request filter.

All queries executed by a user on the data source will then utilize the assigned user-specific login credentials.

For example, the following request filter code assigns the login credentials for a particular user on a JDBC data source named 'Orders':

```
String username = lookupDatabaseUser(principal);
String password = lookupDatabasePassword(principal);
principal.setParameter("_Db_User_Orders", username);
principal.setParameter("_Db_Password_Orders", password);
```

In this example, you would need to create the `lookupDatabaseUser()` and `lookupDatabasePassword()` methods as part of the SSO Filter.

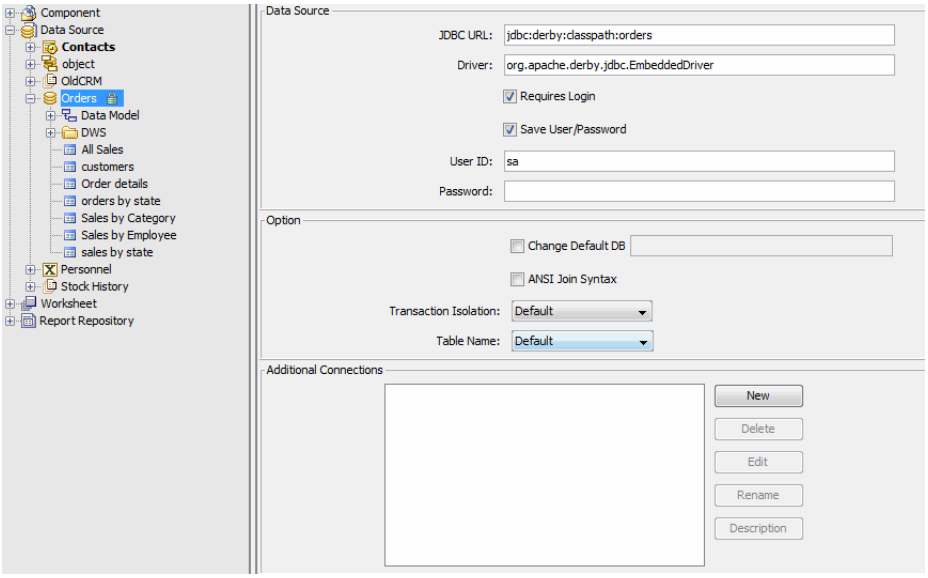
3.1.4 Adding Additional Connections

A single data source can support connections to multiple databases. This is useful in a multi-tenant environment, as it enables you to assign different connections to different users.

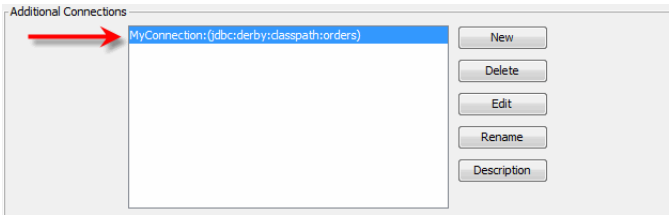
To add a new connection, follow the steps below:

1. From the 'Data Source' node of the **Asset** panel, open (double-click) the data source for which you want to create additional connections.

2. In the 'Additional Connections' panel of the data source definition page, click 'New'. This will start the 'New Data Source' wizard.



3. Follow the steps of the ‘New Data Source’ wizard (see *Creating a New JDBC Data Source* for details), and click ‘Finish’. This adds the new connection to the connection list.



To reopen the ‘New Data Source’ wizard for a connection, select the connection in the list, and click ‘Edit’. To change the connection name or description, click the ‘Rename’ and ‘Description’ buttons, respectively. Click ‘Delete’ to remove the connection.

See Also

Design for Multi-Tenant Environment, for more about multi-tenant considerations.

Creating a New JDBC Data Source, for information on creating the default connection.

Extending a Physical View, to create a physical view based on a new connection.

3.1.5 Connecting to Other JDBC Databases

The examples in the documentation use the sample Derby database (see *Connecting to a Database*). The following sections provide specific instructions for other common JDBC data sources.

SQL Server

Follow the procedure described in *Connecting to a Database*. For SQL Server, the 'JDBC URL' in the JDBC Data Source Wizard has the following form:

```
jdbc:sqlserver://{server name}:1433
```

Note that the SQL Server driver provided with the installation uses SQL Server authentication. You must therefore create a login account within SQL server itself, rather than using a Windows domain account.

See Also
SQL Server Analysis Services, for information about connecting to cubes.

SQL Server Express

Follow the procedure described in *Connecting to a Database*. For SQL Server Express, the 'JDBC URL' has the following form (no port number):

```
jdbc:sqlserver://{server name};instanceName=SQLEXPRESS
```

Use your SQL Server login credentials rather than Windows domain credentials.

Oracle

Follow the procedure described in *Connecting to a Database*. For Oracle, the 'JDBC URL' in the JDBC Data Source Wizard has the following form:

```
jdbc:oracle:thin:@{server name}:1521:{service ID}
```

DB2

Follow the procedure described in *Connecting to a Database*. For DB2, the 'JDBC URL' in the JDBC Data Source Wizard has the following form:

```
jdbc:db2://{server name}:50000/{database name}
```

MySQL

Follow the procedure described in *Connecting to a Database*. For MySQL, the 'JDBC URL' in the JDBC Data Source Wizard has the following form:

```
jdbc:mysql://{server name}:3306/{database name}
```

Microsoft Access

Follow the procedure described in *Connecting to a Database*. For Microsoft Access, the 'JDBC URL' in the JDBC Data Source Wizard has the following form:

```
jdbc:odbc:{DSN}
```

To create an ODBC system DSN (Data Source Name) in Windows, open Windows' *Administration Tools* and select *Data Sources (ODBC)*.

Note that the 64-bit ODBC driver is not pre-installed on Windows. If you are using the default 64-bit version of Style Scope or Style Intelligence, you will need to download the 64-bit ODBC driver from Microsoft's Download Center.

Alternatively, you can use the 32-bit ODBC driver, *odbcad32.exe*, located in *C:\Windows\SysWOW64*. However, in order to use this driver, you will need to install the 32-bit version of Style Scope or Style Intelligence.

3.2 Data Models

Relational database schemas are often designed using normalized forms that are efficient for data storage. Normalization reduces redundancy and helps maintain data integrity. However, normalized designs also make it difficult for non-experts to understand the structure of the database, and require complex queries to retrieve meaningful data. This creates a barrier between business users and the data that they require to make decisions.

To remove this restriction and facilitate greater access to data, Style intelligence provides a data representation called a *data model*. The data model allows you to expose business data to report designers and end users in a way that is easy for them to understand and use. You can create data models for any type of database.

```
Data models are not available for the 'Hadoop Hive' JDBC type.
```

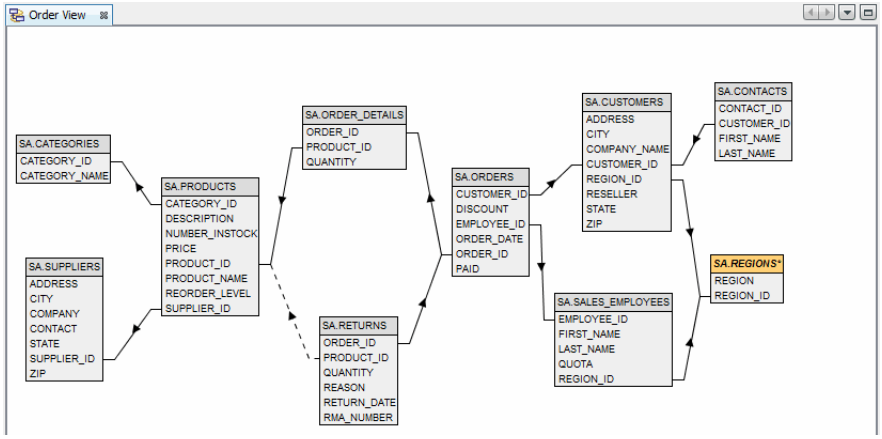
A data model comprises a *physical view*, *logical model*, and optional *hierarchical overlay*. These components are described below:

- **Physical View**: A physical view specifies a subset of database tables and the join relationships between them, covering all or part of the database schema. The physical view transforms the original database schema into a business intelligence-friendly schema, which can then be exposed to designers and end users through one or more logical models. See *Physical View* for information on constructing a physical view.
- **Logical Model**: A logical model is based on a single physical view, and presents a business-oriented, easy-to-understand representation of the physical view by mapping database tables into *entities* and columns into *attributes*. The logical model denormalizes the data (adds additional redundancy) in order to maximize simplicity for data analysts and end users. See *Logical Model* for information on constructing a logical model.
- **Hierarchical Overlay**: A hierarchical model (OLAP overlay) is a mapping that allows OLAP operations to be used on non-OLAP

databases. See *OLAP Overlay – Multidimensional Analysis* for information on constructing a hierarchical model.

3.3 Physical View

The basis of a data model is the *physical view*, which specifies a set of database tables and the joins that relate them. The physical view does not represent a *particular* query that retrieves specific fields, but rather outlines the full join structure that will enable arbitrary queries to be generated on an *ad hoc* basis. When an end-user or report developer accesses a set of attributes in a data model, the joins that you have pre-specified in the physical view will permit the necessary query to be constructed automatically.



A physical view is not directly accessed by report designers or end users. Designers and users have access only to the logical models defined from the physical view. The goal in creating a physical view is therefore to transform the generic database schema into a business-friendly schema, a set of tables and joins to provide the information desired by report designers and end users.

```
If you make a change to the structure of the database schema, press
the 'Refresh Metadata' button on the toolbar to update the physical
view.
```

Physical views are independent of one another; that is, relationships in one physical view do not impact relationships defined in a different physical view. You can create multiple physical views from a given data source, and it is common practice to model a single schema with multiple overlapping physical views. However, a given data model relies on just a single physical view.

3.3.1 Creating a Physical View

To create a new physical view, follow the steps below:

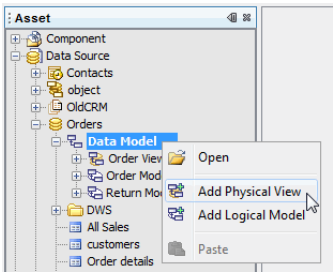
1. *Defining a New Physical View*
2. *Adding Database Tables to the Physical View*
3. *Adding an Embedded SQL View*
4. *Defining Join Relationships*
5. *Resolving Loop Traps*

The following sections explain these steps in greater detail.

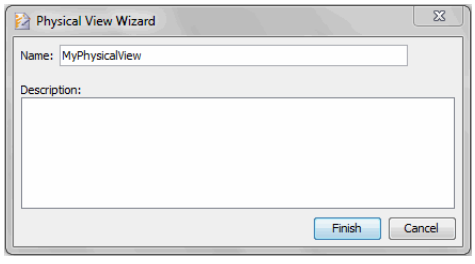
Defining a New Physical View

To define a new physical view, follow the steps below:

1. Click the 'Asset' button to expand the **Asset** panel.
2. Expand the 'Data Source' node, and then expand the relational database data source for which you want to create the data model.
3. Right click the 'Data Model' node, and select 'Add Physical View' from the context menu. This open the 'Physical View Wizard' dialog box.

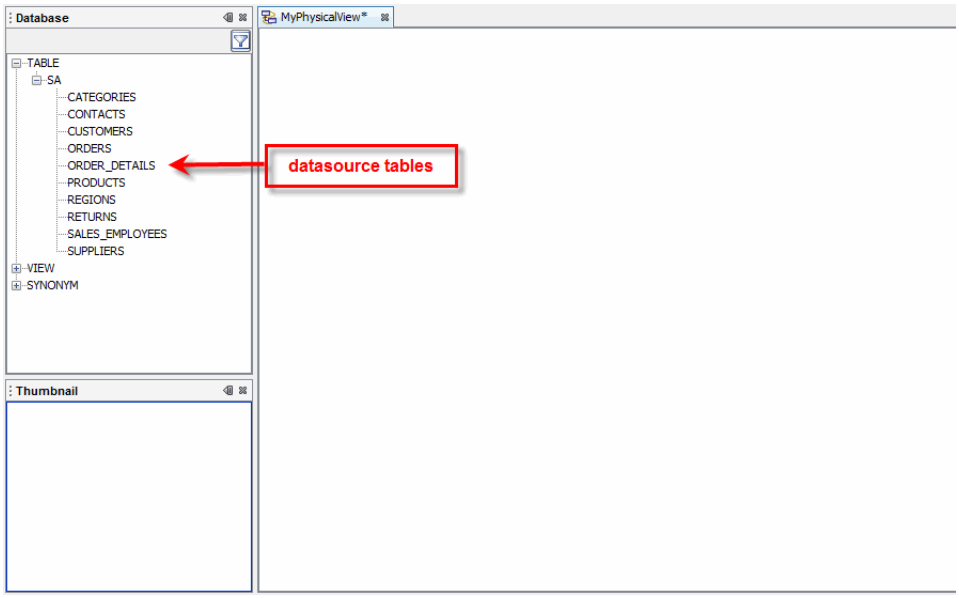


4. In the 'Physical View Wizard' dialog box, enter a name and description for the new view.



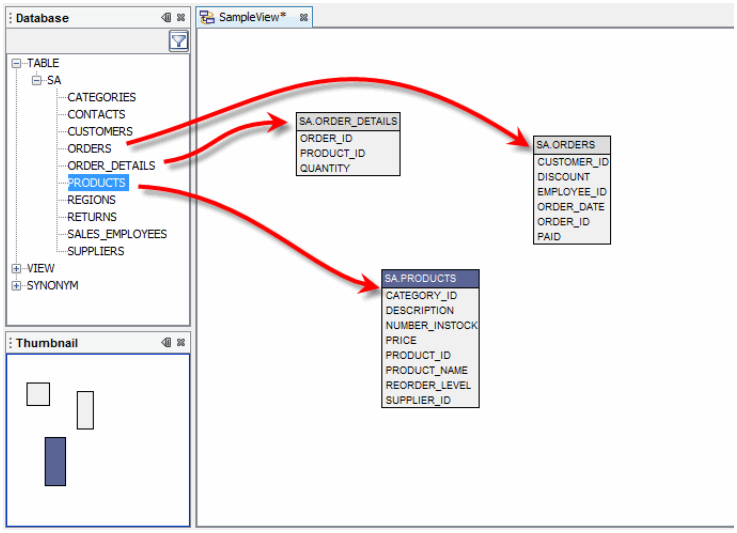
5. Click 'Finish' to close the dialog box.

The name of new physical view is added to the items beneath the 'Data Model' node in the tree, and the new physical view is opened for editing.



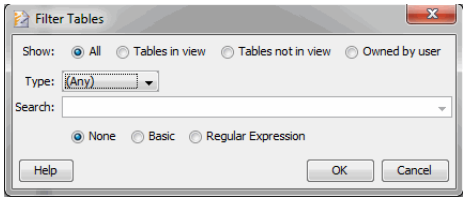
Adding Database Tables to the Physical View

The ‘Database’ panel displays available database tables and views. To add a table to the physical view, simply drag the table from the ‘Database’ panel into the diagram pane (on right).





To filter the list of tables, click the ‘Filter Tables’ button at the top of the ‘Database’ panel. For large databases, this provides a convenient way to quickly find the desired table or view.



Select ‘Owned by user’ to display only those tables that belong to the user’s schema. Select ‘Tables in view’ or ‘Tables not in view’ to list tables that are, or are not, respectively, visible in the diagram pane. Use the ‘Type’ menu to restrict the list to either ‘Tables’ or ‘Views’.

To list only tables that match a specified pattern, select one of ‘Search’ options: ‘Basic’ allows you to use ‘*’ as a wildcard, while ‘Regular Expression’ allows Perl 5 regex syntax.

Adding an Embedded SQL View

If you want to add custom business rules in your model without creating a new view in your database schema, you can create an inline *embedded SQL view*. This allows you to create a virtual “table” by embedding a SQL directly into the physical view.

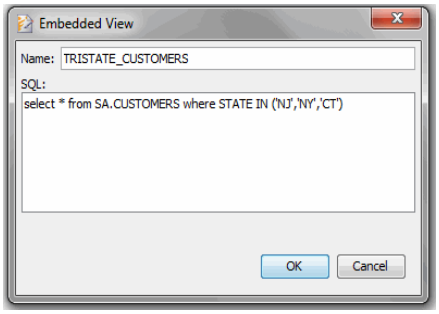
```
Note: An embedded view may result in a different execution plan
than a database view. You should test both options against your
database.
```

To create an embedded view, follow the steps below.

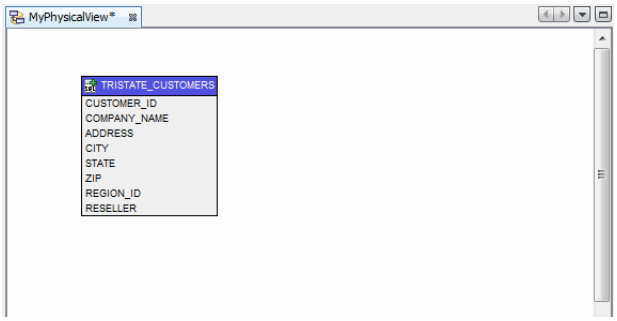


1. Click the ‘New View’ button.

2. Enter the view name in the ‘Name’ field, and type in the SQL query string into the ‘SQL’ field.



3. Press ‘OK’. This adds the embedded view as a new table in the diagram panel.



4. Join this table to other tables/views in the physical view as required.

See Also

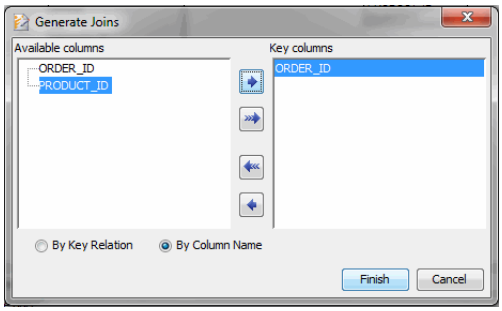
Defining Expression Attributes, for information on creating formula columns.

Defining Join Relationships

You can create join relationships between tables automatically, or by manually joining particular table columns.

To automatically join tables, follow the steps below:

1. Click the 'Auto Join' button in the toolbar. This opens the 'Generate Joins' dialog box.



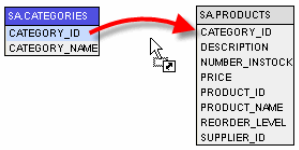
2. Select 'By Key Relation' to join columns by using primary key/ foreign key definitions, or 'By Column Name' to join columns by using common column names.

3. Select the desired columns in the 'Available Columns' panel.

4. Click the right-arrow button to add the selected columns to the 'Key Columns' panel.

5. Click 'Finish' to automatically add the requested joins to the physical view.

To manually define a join between two tables, click a column name in the first table, and drag and drop the column name onto the corresponding column in the second table.



See Also

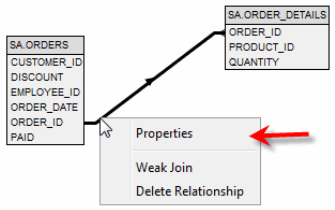
Defining the Join Type, for information on modifying a join.

Defining the Join Type

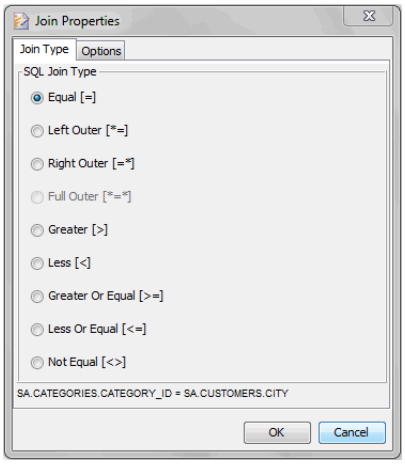
The default join type is an inner join. To change the join type, follow the steps below:

1. Click on the desired join link to select the join.

2. Right-click on the join line, and select ‘Properties’ from the context menu. This opens the ‘Join Properties’ dialog box.



3. Select the desired join type under the ‘Join Type’ tab.



Changing the Join Priority

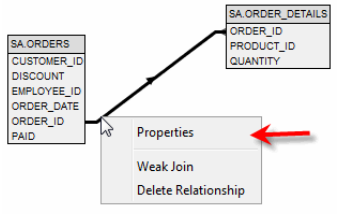
In many cases, the position of a join within a SQL query can greatly affect query performance. To determine the positioning of a particular join within the SQL query generated by the physical view, you can specify a priority for the join.

```
Priority increases with value: 1=lowest, 10=highest. A join with
priority 10 is placed before a join with priority 1.
```

To specify the priority of a join, follow these steps:

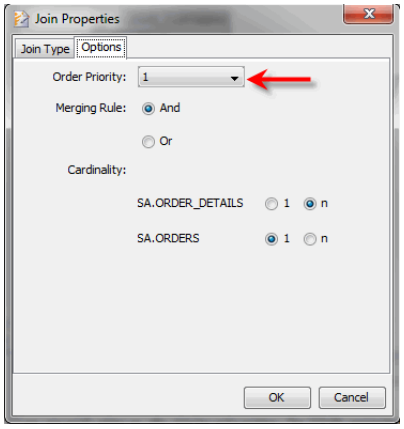
1. Click on the desired join line to select the join.

2. Right-click the join, and select ‘Properties’ from the context menu. This opens the ‘Join Properties’ dialog box.



3. In the ‘Join Properties’ dialog box, click the **Options** tab.

4. From the ‘Order Priority’ menu, select the desired priority for the join.



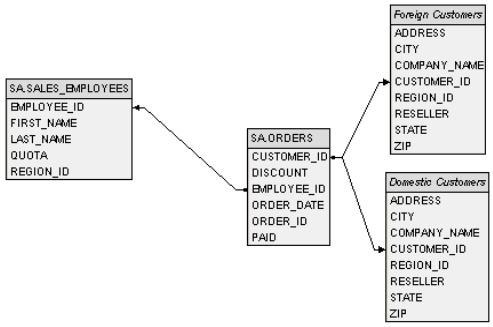
Changing the Join Operator

The default logical operator for joins is ‘AND’. However, there may be some instances where an ‘OR’ operator is appropriate. As an example, consider the following hypothetical case of a Sales schema where an order

can be placed by a domestic customer or a foreign customer. The schema contains three tables:

- Orders
- Domestic Customers
- Foreign Customers

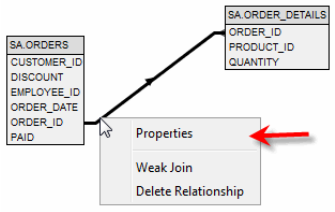
The `CUSTOMER_ID` field in the 'Orders' table can reference the `CUSTOMER_ID` field in either the 'Domestic Customers' or the 'Foreign Customers' table. In this case you need to a logical OR between the "Order -> Domestic Customers" join and the "Order -> Foreign Customers" join because they are mutually exclusive.



To specify the logical operator for a join, follow these steps:

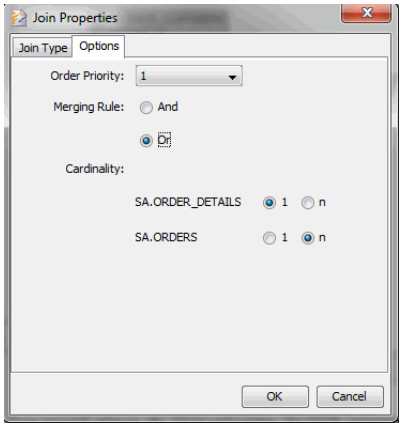
1. Click on the desired join line to select the join.

2. Right-click the join, and select 'Properties' from the context menu. This opens the 'Join Properties' dialog box.



3. In the 'Join Properties' dialog box, click the **Options** tab.

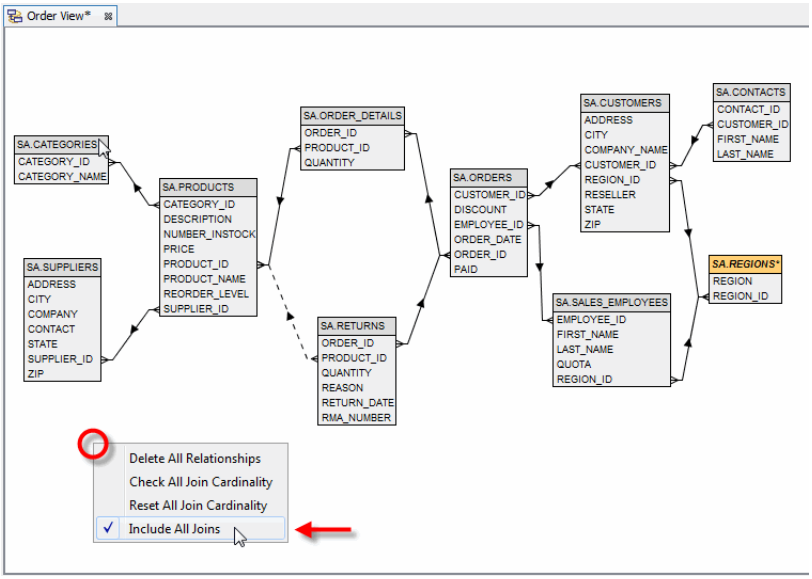
4. Select the desired logical operator in the 'Merging Rule' section, and click 'OK'.



The 'Merging Rule' determines how the join is integrated together with other joins between the same tables. The 'And' option indicates that the join is applied conjunctively (i.e., the join condition must be satisfied even if another join condition is already satisfied); the 'Or' option indicates that the join is applied 'disjunctively' (i.e., the join condition need not be satisfied if another join condition is already satisfied).

Including All Joins in Queries

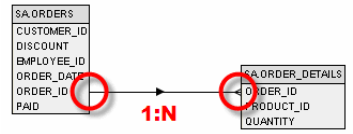
Generally, when Style Intelligence generates a query from a physical view, only the necessary joins between requested tables are included in the SQL string. If you want to include all of the physical view joins in the SQL queries generated from the physical view, right-click in an empty area of the physical view, and select 'Include All Joins' from the context menu.



Overriding Automatic Cardinality Detection

As discussed in *Identifying Query Traps*, Style Studio automatically detects query traps known as the chasm trap and the fan trap by analyzing the cardinality and connectivity of relationships between tables. As a result, Style Studio is able to generate an appropriate warning message when a report developer or end user constructs a data model binding that creates a query trap. Because Style Studio generates these warnings automatically, you do not need to manually analyze your model for query traps in general.

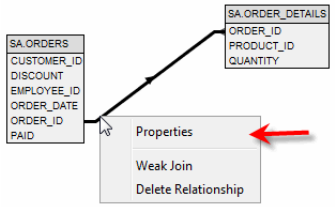
The automatically-detected cardinality for a join is indicated by a fan icon on the join link. However, if you need to override the automatic cardinality detection, you can do this by manually specifying the relationship between tables associated by a join.



To manually specify the cardinality relationship for a join, follow the steps below:

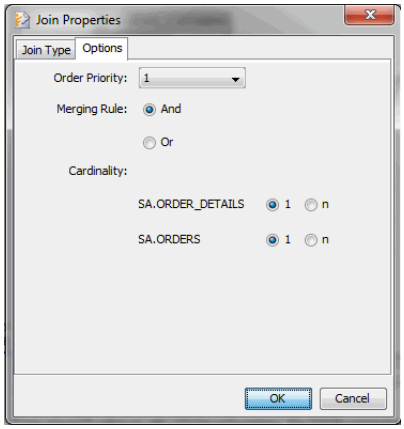
1. Click on the desired join line to select the join.

2. Right-click the join, and select ‘Properties’ from the context menu. This opens the ‘Join Properties’ dialog box.



3. In the ‘Join Properties’ dialog box, click the **Options** tab.

4. Select the desired cardinality relationship in the ‘Cardinality’ section, and click ‘OK’.



The ‘fan’ icons on the ends of the join links will update to reflect the cardinality relationship that you have specified.

Resolving Loop Traps

Style Studio alerts you to loop traps by highlighting the implicated joins in bright red. There are various approaches to resolving loop traps, described below.

- *Weak join* designates a relationship to be used only when no other join route is available. Right-click on a relation line and select ‘Weak Join’ from the menu to mark a join as such. Weak joins appear as dotted lines.

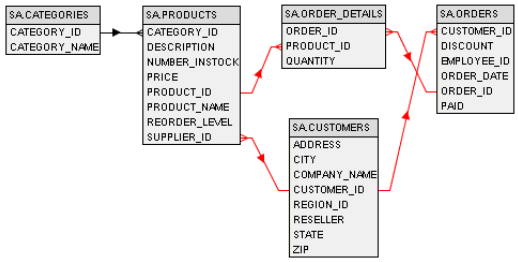
```
An aliased table is indicated by a colored header. To display the
name of the table that was originally aliased, hover the mouse over
the alias table header.
```

- *Manually aliasing* a table is another tool in resolving traps. Select a table and right-click to select the ‘Create Alias’ option. Once the alias is created, it can be used as a totally independent copy of the original table.
- *Auto aliasing* provides a new level of flexibility. Creating alias tables manually is very simple and straightforward when the aliased table stands alone. However if a table has outgoing relationship links, aliasing the table may mandate further aliasing of other tables. The auto-aliasing feature significantly simplifies this process.

Example:
Aliasing a Table

Consider a simple example in which a table called ‘PRODUCTS’ needs to be aliased in order to resolve a cycle. Follow the steps below:

1. Create a new physical view, as shown below.



The joins highlighted in red indicate the presence of a cycle. You can resolve the cycle by auto-aliasing the ‘PRODUCTS’ table, as the following steps illustrate.

2. Right-click on the title of the ‘PRODUCTS’ table and select ‘Auto-alias’ from the context menu. This opens the ‘Auto-alias’ dialog box.

3. In the ‘Auto-alias’ dialog box, check the ‘Enable Auto Aliasing’ box. The dialog box lists all of the joins from the adjacent tables in the diagram.

4. In the ‘Incoming Joins’ panel, select the boxes next to the ‘ORDER_DETAILS’ and ‘CUSTOMERS’ tables.

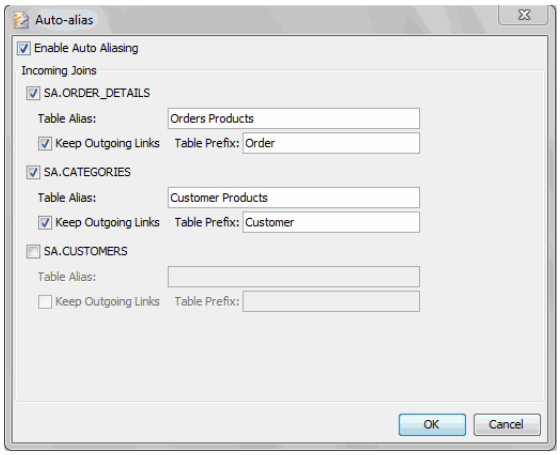
   This designates the ‘ORDER_DETAILS’ and ‘CUSTOMERS tables as providing the “incoming” joins to the ‘PRODUCTS’ table. When the ‘PRODUCTS’ table is auto-aliased, two copies of the table will be created. One copy corresponds to the incoming join from the ‘ORDER_DETAILS’ table, and the other copy corresponds to the incoming join from the ‘CUSTOMERS’. By splitting the ‘PRODUCTS’ table into two aliases, the join cycle is eliminated. (Note, however, that if only the ‘PRODUCTS’ table is aliased, the cycle will simply reappear on the ‘CATEGORIES’ table. This is why in the succeeding steps, you will specify the ‘Keep Outgoing Links’ option.)

5. In the ‘Table Alias’ field for the ‘ORDER_DETAILS’ table, enter “Order Products”.

6. Select ‘Keep Outgoing Links’ for the ‘ORDER_DETAILS’ table, and enter “Order” as the ‘Table Prefix’. This will cause the downstream ‘CATEGORIES’ table to be aliased for the ‘ORDER_DETAILS’ join path as well.

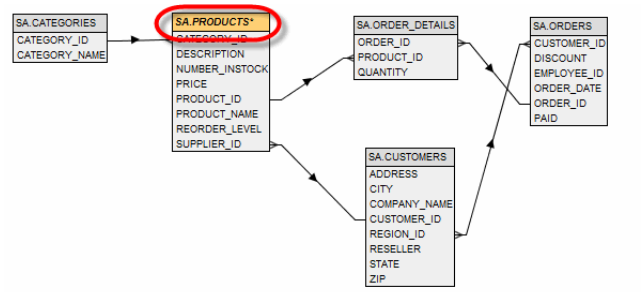
7. In the ‘Table Alias’ field for the ‘CUSTOMERS’ table, enter “Customer Products”.

8. Select ‘Keep Outgoing Links’ for the ‘CUSTOMERS’ table, and enter “Customer” as the ‘Table Prefix’. This will cause the downstream ‘CATEGORIES’ table to be aliased for the ‘CUSTOMERS’ join path as well.

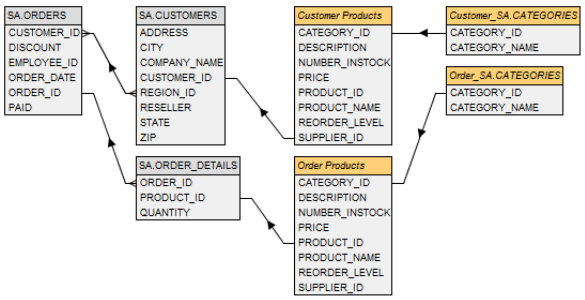


Note: In the diagram, an *italicized* table name and colored header indicate that the table is aliased. To view the full effect of aliasing, however, you must preview the physical view (below).

9. Click ‘OK’ to complete the auto-alias procedure. Note the italics and color on the ‘PRODUCTS’ table, indicating aliasing.



10. Click the ‘Preview’ button in the Style Studio toolbar. This presents the structure of the physical view as it will *actually* be used for generating queries.



Observe that both the ‘PRODUCTS’ table and the ‘CATEGORIES’ table have been aliased, and that this has entirely eliminated the cycle.

The new tables that appear in the preview (*Customer Products*, *Order Products*, *Customer_SA.CATEGORIES*, *Order_SA.CATEGORIES*) have been introduced into the physical view by aliasing. They will appear among the tables available for constructing the *Logical Model*.

With the elimination of the cycle, there is no remaining ambiguity in the join structure; any set of fields that one selects from the various tables has a single unique join path. However, when you design the logical model, you must still be careful when selecting fields to expose as model attributes. For example, ‘Customer Produsts.PRODUCT_NAME’ will in general return different results than ‘Order Produsts.PRODUCT_NAME’ when these fields are used in a data model. (This is because, even though these two tables are identical aliases, they are joined to the other tables in different ways, which results in different data selection.)

¢

See Also
Identifying Query Traps, for information about loop traps and other traps.

3.3.2 Navigating a Physical View



To automatically organize a physical view diagram, click ‘Auto Layout’ in the toolbar. This repositions the tables to improve the clarity of the diagram. (It does not change the join structure or join properties.)



To find a particular table within a complex diagram, enter the beginning of the table name in the toolbar’s ‘Find’ box. Then click the ‘Find Table’ button. This will select all tables that match the specified prefix.

3.3.3 Identifying Query Traps

A *query trap* refers to a join construction in the physical view that generates query results that might confuse or mislead end users. There are three basic kinds of query traps: *loop traps*, *fan traps*, and *chasm traps*. The following sections discuss these trap types in greater detail.

Star schemas are generally free from query traps because of their uniform 1:n relationship from dimension table to fact table, and location of all measures in fact table.

Style Intelligence automatically detects all of these types of query traps. To avoid fan traps and chasm traps, Style Intelligence detects the cardinality of join relationships, and warns users and developers when they select a subset of data model attributes that would instantiate a potential query trap. In Ad Hoc report creation, attributes that would instantiate a query trap are rendered in grey, indicating that they should not be selected.

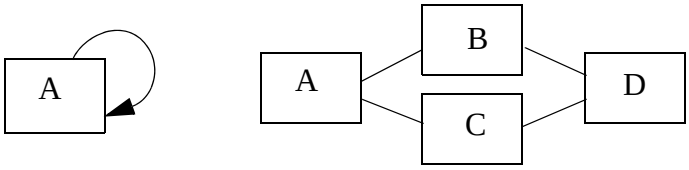
In general, because fan and chasm trap detection is handled automatically, you should not need to make explicit modifications. However, if the automatic cardinality detection incorrectly identifies a trap where none exists (which might generate unnecessary warnings for end users), you can override the default cardinality detection through the 'Join Properties' dialog box. See *Defining Join Relationships* for details.

For loop traps, Style Intelligence highlights the offending joins in the physical view using red color. Loop traps are not corrected automatically, and you should select the appropriate method for resolving loops based on your knowledge of the schema. See *Resolving Loop Traps* for full details.

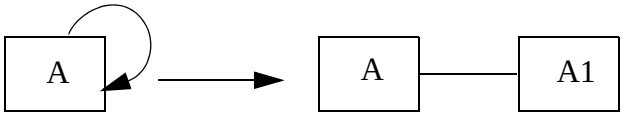
Additionally, you will be alerted to the existence of un-joined orphan tables when you save the physical view. In general, you should make sure that every table in a physical view is joined to at least one other table. However, when you are prompted with the warning message, you can choose to force the physical view to be saved with orphaned tables. To correct the view, select 'Cancel' in the warning message, and Style Studio will highlight the detached tables.

Loop Traps

A *loop* or *cycle* is present in the physical view of a data model when multiple possible paths exist from table A to table B. A loop can exist as a self-join or multiple join path. Style Studio automatically highlights these loops in red.



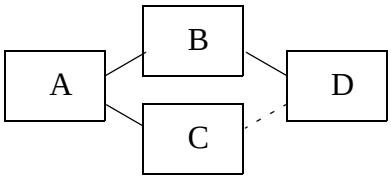
For example, when an "Employee" table contains a "manager id" is joined to itself (i.e., to indicate an employee's manager), this produces a self-join loop. You must resolve the self-join to allow end users to find employees and their respective managers. Self-join loops are most easily resolved by creating a table alias. (See *Resolving Loop Traps* for details.)



A multiple-table loop has the potential to create results that are too restrictive. For example, consider an order table (A) that joins to the "ship to" table (B) and the "bill to" table (C), which in turn joins to a "company"

table (D). If end users select fields from all four tables, only orders that have the same company for “bill to” and “ship to” will be returned.

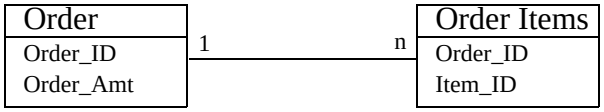
Multi-table loops can be addressed using aliasing or auto-aliasing. Additionally, these loops can be addressed by using a weak join which will not be included in the join path of the resultant query. (See *Resolving Loop Traps* for details.)



Fan Trap

A *measure* is a numeric field that can be aggregated. Aggregation of fields at the detail level is always acceptable, but incorrect results can be produced when a summary-level fields are aggregated. The problem of these higher-level aggregates is known as the *fan trap*.

For example, consider an “Order” table containing total order amounts. If this table is joined with an “Order Items” table, then aggregation by order item will recount the order amount multiple times if an order contains more than one item.



This trap can be readily identified by relationship cardinality (indicated by the “fan” symbol on the join terminus). If the specified measure exists in the “1” side table that has a **1:n** relationship with another table, this trap will occur.

The *fan trap* is evidence of an inherent deficiency in this particular schema, wherein the schema is not providing the order amount on the item-level of granularity. If this aggregation is desired, the best option is to enhance the schema so that the order amount is broken down and recorded in the ‘Order Items’ table.

If this aggregation is not desired, use two physical views to separate the measure and the lower-granularity table to prevent incorrect aggregation. You can still create a data view that shows both order price and all associated items by building the desired Data Block within a *Advanced Data Mashup*.

Chasm Trap (Cardinality Trap)

When two tables are related by way of a higher-level table (representing a "compression"), information returned by queries on these tables can be misleading. This problem of a higher-level join is known as the "chasm trap."

For example, a "Customer" table is related to a "Product" table to record products ordered by a customer. The same "Customer" table is also related to support case "Issue" table that records the product issues reported by the customer.



This physical view presents the appearance that it is possible to view issues by product. However, because the "Product" table and "Issue" table are linked through a higher-level table ("Customer") that preserves only individual customers, this is in fact not possible. Through these particular joins, each product a customer purchased is associated with *all* issues this customer raised – regardless of the particular product.

This trap is readily identifiable by the combination of **n:1** and **1:n** join cardinalities. If you require "issue by product" data, you can enhance the physical view by relating the "Issue" table directly to the "Product" table, and resolving the resulting loop with a weak join. (See *Resolving Loop Traps*.) Otherwise, you should isolate the two relationships in two separate physical views. You can still combine the two views by building the desired Data Block within a *Advanced Data Mashup*.

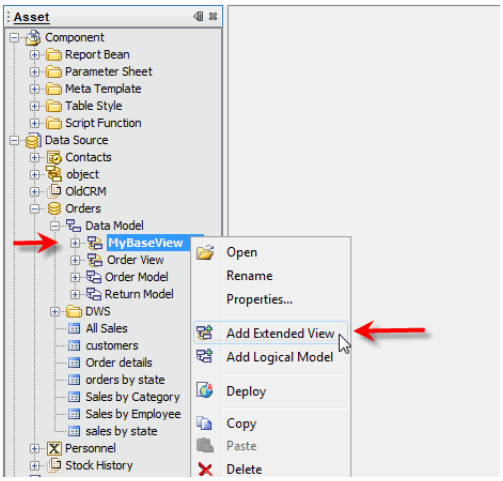
3.3.4 Extending a Physical View

In a multi-tenant environment, it may be useful to provide a different physical view to each tenant. Each physical view can provide access to tenant-specific database tables, as well as to a set of common tables that are shared by all physical views.

To create a tenant-specific physical view, extend the existing physical view by adding tables from a tenant-specific connection. Follow the steps below.

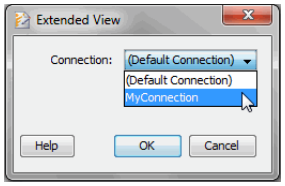
1. Open the **Asset** panel, and expand the 'Data Source' node.

2. Right click on the base physical view that you wish to extend, and select 'Add Extended View' from the context menu. This opens the 'Extended View' dialog box.



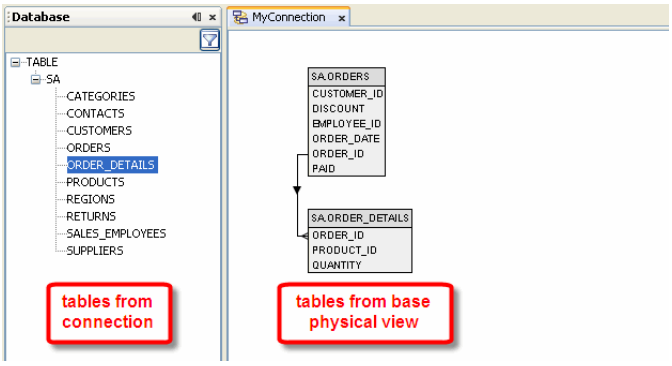
Note: An extended physical view is associated with one (and only one) connection. Likewise, a connection can be associated with only one extended physical view.

3. In the 'Extended View' dialog box, choose a connection option for the new extended view, and click 'OK'. (You can select the default data source connection, or any of the tenant-specific connections.)



Tables and joins inherited from the base physical view cannot be modified within the extended view.

This adds the new extended view to the current data source, and opens the view for editing. The right panel displays the tables and joins defined in the base physical view.



4. Drag tables from the left panel into the right panel, and create any desired joins between the new tables and the inherited tables. (See *Creating a Physical View* for more information about joining tables in a physical view.)

5. When you have finished defining the extended view, click the ‘Save’ button in the Style Studio toolbar.

You can now use this extended physical view to create an extended logical model. See *Extending a Logical Model* for more information.

See Also
Adding Additional Connections, for information on configuring connections.
Creating a Physical View, for information on creating the base physical view.

3.4 Logical Model

A physical view can only be exposed to end users through a logical model.

A *logical model* presents a business-oriented, easy-to-understand view of a single underlying physical view. The logical model organizes data from the physical view into logical *entities* that correspond to business world objects, rather than to the underlying physical tables. Each entity contains one or more *attributes* that map to database table columns from the physical view. Logical models can also extend existing database fields using formula (expression) attributes.

The mapping of tables and fields into *entities* and *attributes* is not a simple regrouping. Rather, the logical model *denormalizes* or adds redundancy to the data, in order to maximize simplicity for designers and end users.

3.4.1 Creating a Logical Model

To create a new logical model, follow the steps below:

1. *Defining a New Logical Model*
2. *Adding Entities and Attributes*
3. *Modifying Entities and Attributes*
4. *Defining Expression Attributes*

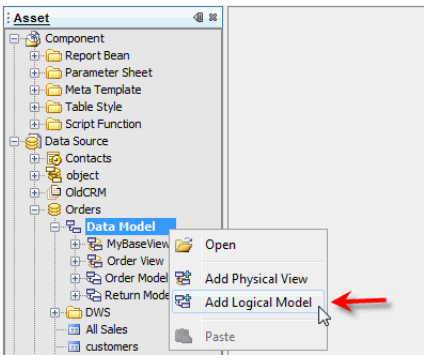
The following sections explain these steps in greater detail.

Defining a New Logical Model

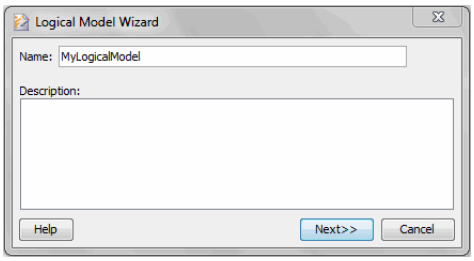
To define a new logical view, follow the steps below:

1. Click the ‘Asset’ button to expand the **Asset** panel.
2.  Expand the ‘Data Source’ node, and then expand the relational database data source for which you want to create the data model.

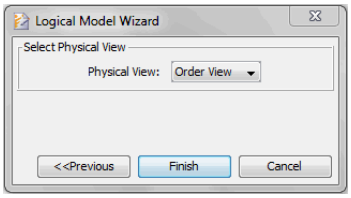
3. Right click the ‘Data Model’ node, and select ‘Add Logical Model’ from the context menu. This open the ‘Logical Model Wizard’ dialog box.



4. In the ‘Logical Model Wizard’ dialog box, enter a name and description for the new model. Click ‘Next’ to proceed.

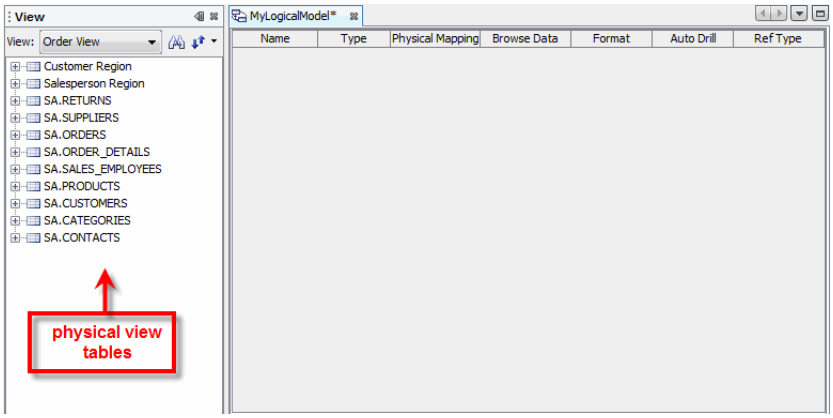


5. From the ‘Physical View’ menu, select the physical view on which the model should be based. (The model will have access to the tables of the underlying physical view.)



6. Click ‘Finish’ to close the dialog box.

The name of new logical model is added beneath the ‘Data Model’ node in the tree, and the new logical model is opened for editing.

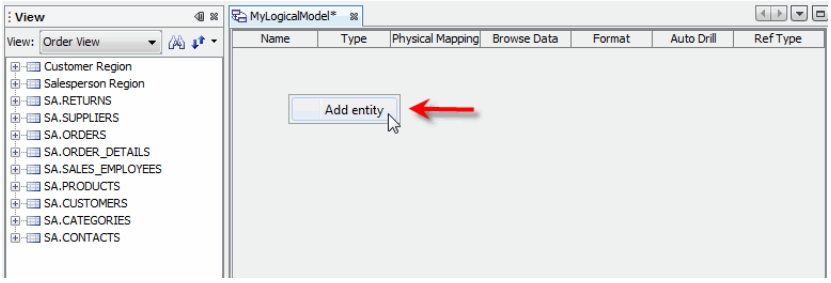


Adding Entities and Attributes

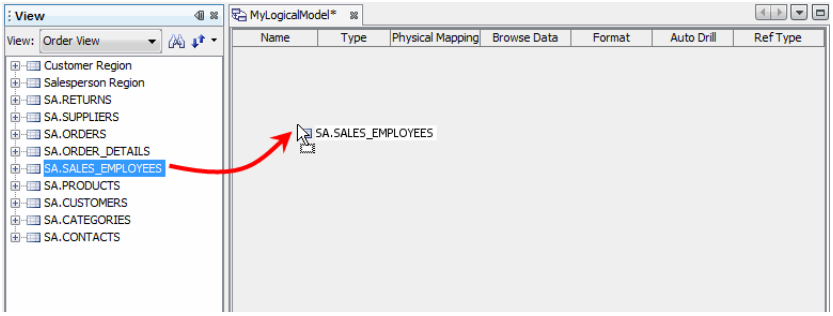
An *entity* is a table-like construct that contain one or more attributes. An *attribute*, in turn, represents a single field from the database. You will use entities and attributes to create a business-friendly view of the database schema.

To create a new entity, use one of the following methods:

1. Right-click an empty region of the logical model panel, and select 'New Entity' from the context menu. This opens the 'Entity Wizard', where you can enter a name for the new entity.

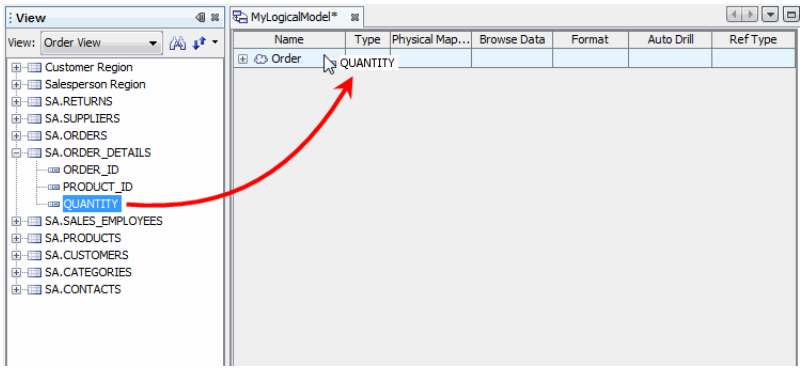


2. Drag a table from the physical view panel into the logical model panel. This adds a new entity containing all of the table fields as attributes. (Expand the entity node to see the attributes.)



To search for a table, see *Sorting and Filtering a Physical View*.

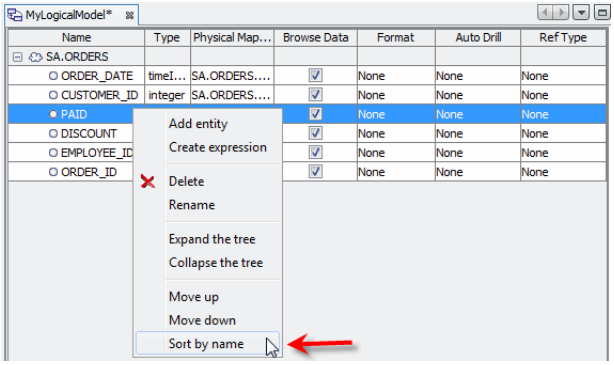
To add an attribute to an entity, drag a field from the **View** panel onto the desired entity. (Expand the entity node to see the component attributes.)



Modifying Entities and Attributes

You can modify entities and attributes in a variety of different ways, as described below:

- **Moving Entities/Attributes**: To move an entity or attribute from one location to another (either within or between entities) simply drag and drop the entity/attribute in the desired location.
- **Sorting Entities/Attributes**: To sort entities or attributes alphabetically, right-click on the desired entity/attribute, and select 'Sort by name' from the context menu.



- **Renaming Entities/Attributes**: To rename an entity or attribute, double-click the desired entity/attribute. This opens an edit box in which you can modify the name. Entity names cannot contain two consecutive colons, "::", because this is used internally as a delimiter.
- **Deleting Entities/Attributes**: To delete an entity or attribute, click to select the entity or attribute or Ctrl-click to select multiple objects. Then right-click on a selected entity/attribute, and choose 'Delete' from the context menu. (You can also press the 'Delete' key on the keyboard.)

- **Enabling Data Browsing**: To allow designers and users to browse the attribute values when they set a filtering condition on the attribute (for example, see *Creating a Filter Condition* in *Advanced Dashboard Design*), enable the corresponding 'Browse Data' option. By default, 'Browse Data' allows designers and users to select from a list of the distinct values contained in the first 1000 rows. To provide a customized list of values to users see *Customizing the Browse Data List*.
- **Adding Auto-Drilldown to Attributes**: To add automatic hyperlinks to an attribute, see *Auto-Drilldown*.
- **Adding Formatting to Attributes**: To add formatting to the data returned in an attribute, see *Adding a Format to a Data Field*.
- **Adding Reference Type to Attributes**: To specify that a field should be treated by default as a dimension, select 'Dimension' from the 'Ref Type' menu. To specify that a field should be treated by default as a measure, select the desired aggregation method from the 'Measure' folder in the 'Ref Type' menu. The 'Ref Type' setting determines the attribute's default behavior (as dimension or measure) in aggregation contexts such as chart and table bindings, etc.

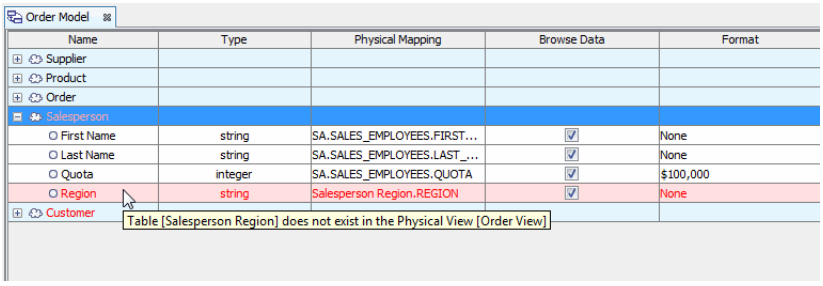
Detecting Model Validity Problems

If you make changes to the underlying database schema or to the physical view (for example, if you remove tables or columns), the mappings in the logical model may no longer be valid. When Style Studio detects a validity problem in the field mappings, it highlights the invalid entities and attributes in red.

To determine the cause of a validity failure, follow the steps below:

1. Press the 'Refresh Metadata' button in the toolbar to verify that Style Studio has access to the current schema information.

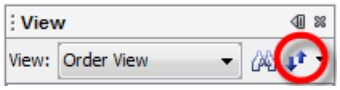
2. In the logical model, hover the mouse over the 'Name' field or 'Physical Mapping' field of the red-highlighted attribute. This displays a message explaining the cause of the mapping problem.



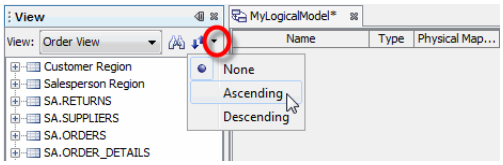
3. When you have determined the source of the problem, adjust the database schema, physical view, or logical model to restore a valid mapping.

Sorting and Filtering a Physical View

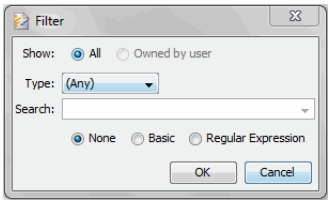
When you add table fields from a physical view to a logical model, it is often helpful to sort the physical view tables by name. To do this, click the  'Sort Tables' button on the **View** panel.



Click the button once to sort the tables in ascending order. Click again to sort in descending order. Click a third time to return to the original order. To jump to a specific sort order without cycling through all the options, click  the right side of the button to expand the menu.



To quickly find tables of interest, you can also filter the list. To do this, click  the 'Filter Visible Tables' button on the toolbar. This opens the 'Filter' dialog box.



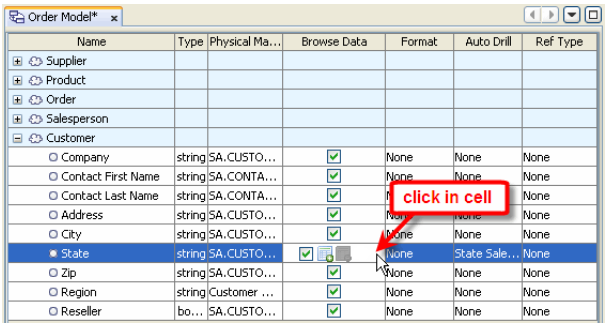
Select 'Owned by user' to display only those tables that belong to the user's schema. Use the 'Type' menu to restrict the list to either 'Tables' or 'Views'.

To list only tables that match a specified pattern, select one of 'Search' options: 'Basic' allows you to use '*' as a wildcard, while 'Regular Expression' allows Perl 5 regex syntax.

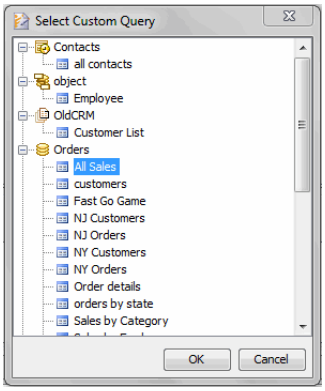
Customizing the Browse Data List

To provide a customized list of values to users who select the 'Browse Data' option in a condition, specify a customized query. To do this, follow the steps below:

1. Click inside the ‘Browse Data’ column corresponding to the desired attribute. This opens a mini-toolbar in that cell.



2. Click the ‘Select custom query’ button. This opens the ‘Select Custom Query’ dialog box.



3. Select the desired query.

   The first column of this query should return the desired attribute *values*, i.e., the values that will actually be sent to the database when filtering the data. The second column of the query should contain the corresponding *labels*, i.e., the text that will be shown to users in the ‘Browse Data’ menu. Query parameters, if required, can be specified as HTTP session attributes. (See *Session-Based Single Sign-On* in *Advanced Server Integration* for additional information). If a required query parameter is not specified, the corresponding condition is dropped.

4. Click ‘OK’ to close the ‘Select Custom Query’ dialog box.

5. Click the ‘Apply’ button in the mini-toolbar (or simply click away) to finalize the settings.

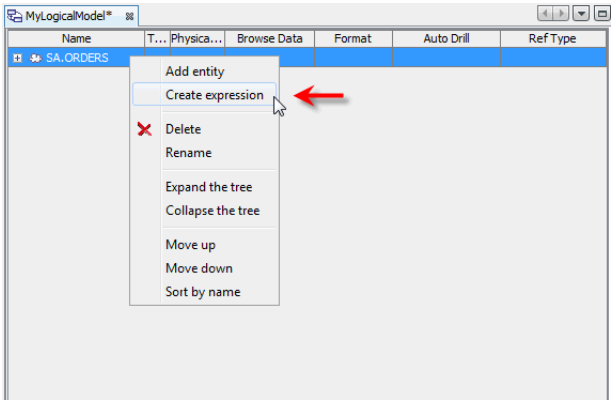
The selected query will now be used to supply the list of values that appears in the ‘Browse Data’ list for the corresponding attribute.

Defining Expression Attributes

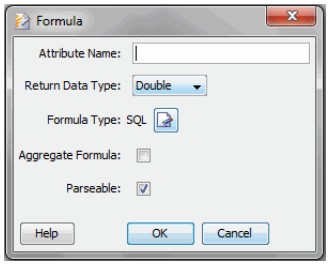
In addition to creating model attributes by directly mapping fields of the physical view, you can also define *expression attributes*. Expression attributes are attributes that are *derived* from table columns. You can use derived attributes in the same way as any other model attribute, e.g., in reports, worksheets, hierarchical overlays, etc. (See *Design Considerations* for more information on hierarchy).

To define an expression attribute, follow the steps below:

1. Right-click on an entity or attribute in the logical model, and select 'Create expression' from the context menu.



This opens the 'Formula' dialog box.



2. In the 'Attribute Name' field of the 'Formula' dialog box, specify a name for the new attribute.

3. In the 'Return Data Type' menu, select the data type of the value returned by the defined expression.



4. Click the 'Edit' button to open the Script Editor.

5. Enter the expression to define the new attribute. (No `return` keyword is necessary.) The expression should use the appropriate SQL syntax for the underlying database.

6. Click ‘Check’ if you want Style Studio to check the syntax of the specified SQL expression.



7. Click the ‘Save and Close’ button.

8. Select ‘Aggregate Formula’ if the specified expression contains aggregation functions such as `COUNT`, `AVG`, `SUM`, etc., or other database-specific aggregation functions. This will ensure that generated queries include an appropriate `GROUP BY` clause.

9. Select ‘Parseable’ if you want VPMs to be able to hide this field based on its component fields. For example: a VPM hides the ‘price’ column for certain roles, and an expression ‘revenue’ is defined as ‘price’ multiplied by ‘quantity’. The ‘revenue’ column will also be hidden for those same roles, as long as it is marked ‘Parseable’.

10. Click ‘OK’ to exit the ‘Formula’ dialog box. The new expression attribute will appear alongside the existing attributes under the selected entity.

Example: Defining Expression Attributes

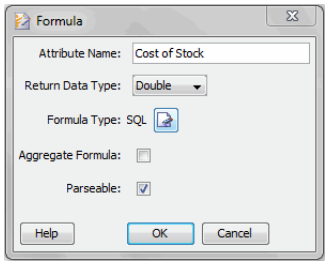
In this example, you will add a new expression attribute to the sample Order Model. The attribute will represent the total price of a particular product’s stock (i.e., product price * number in stock).

1. Open the **Asset** panel. Expand the ‘Data Source’ node and the ‘Orders’ data source.

2. Double-click on the ‘Order Model’ to open the model for editing.

3. Right-click the ‘Product’ entity in the logical model, and select ‘Create expression’ from the context menu. This opens the ‘Formula’ dialog box.

4. In the ‘Attribute Name’ field, enter “Cost of Stock” as the name of the new attribute.





5. Click the Edit button to open the Script Editor, and enter the following expression:

```
field['SA.PRODUCTS.PRICE'] *
field['SA.PRODUCTS.NUMBER_INSTOCK']
```

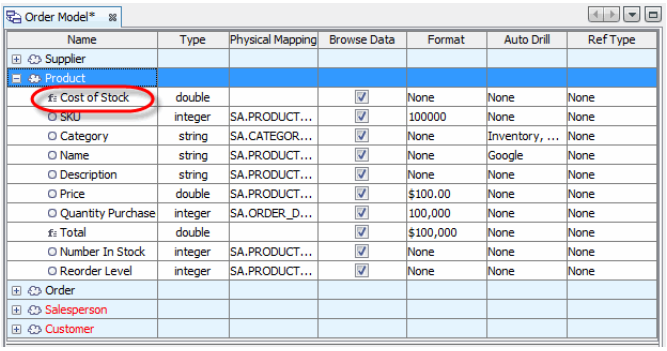
Note: You can double-click the field names in the ‘Properties’ list to add them to the expression (without typing).



6. Click the ‘Save and Close’ button.

7. Select ‘Double’ or ‘Float’ from the ‘Return Data Type’ menu.

8. Click ‘OK’ to exit the ‘Formula’ dialog box.



The new expression attribute ‘Cost of Stock’ now appears alongside the other attributes of the ‘Product’ entity. It is marked with an ‘f’ symbol, which indicates a formula field.

To modify an existing expression attribute, right-click the attribute and select ‘Edit expression’ from the context menu. This opens the ‘Formula’ dialog box, where you can make the desired modifications as discussed above.

¢

3.4.2 Auto-Drilldown

You can also set an auto-drill for a particular level of a cube dimension.

You can assign hyperlinks to data model attributes or query columns at the time that you design the model or query. Hyperlinks can target reports, Viewsheets, or web pages. When you use the model or query within a report or Viewsheet, the hyperlinks allow the user to drill down to the specified reports, Viewsheets, or web pages. This is known as *auto-drilldown*, because the specified hyperlinks are automatically created wherever the model or query is used.

You can pass parameters with the drill-down hyperlinks. Parameter values can include the value of the attribute/column on which the user clicks to launch the hyperlink, or mapped values obtained from a distinct query.

The advantages of using auto-drilldown (rather than using regular hyperlinks at the report or Viewsheet level) are that drilldown parameters can incorporate the results of arbitrary queries, and that the drilldowns are automatically propagated to all reports and viewsheets that use the model or query.

See Also
Hyperlinks, in *Advanced Report Design*, to add explicit links directly to a report element.
Hyperlinks, in *Advanced Dashboard Design*, to add explicit links directly to a Viewsheet.

Adding Auto-Drilldown to a Logical Model

The following examples illustrate how to pass parameters to a target report or web page by adding an auto-drilldown to a logical model. There are two basic kinds of parameter values that you can pass in an auto-drill:

- You can pass actual column data values, i.e., the values that the end user clicks to launch the hyperlinks.
- You can pass values derived from a query based on the column data.

Example 1: Passing column values in a drill

Set your Style Studio repository directory to the server SREE home (*WEB-INF/classes*). See *Configuring a Repository* for more information.

In this example you will add an auto-drill to the "Order Model" to provide drilldown hyperlinks from the 'Customer' > 'Company' attribute to the 'OrderList' report (in *classes/inetsoft/demos*). The 'OrderList' report is designed to accept five parameters, one of those being 'customer,' which represents the company name of the customer.

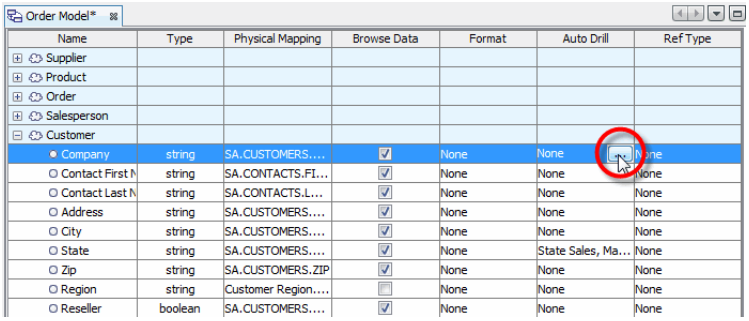
Walkthrough

You will configure an auto-drill on the 'Customer' > 'Company' attribute of 'Order Model' to pass the actual company name (column value) into the 'OrderList' report's 'customer' parameter. (For an illustration of how to target a web page rather than a report, see *Example 2: Passing query-based values in drill*.)

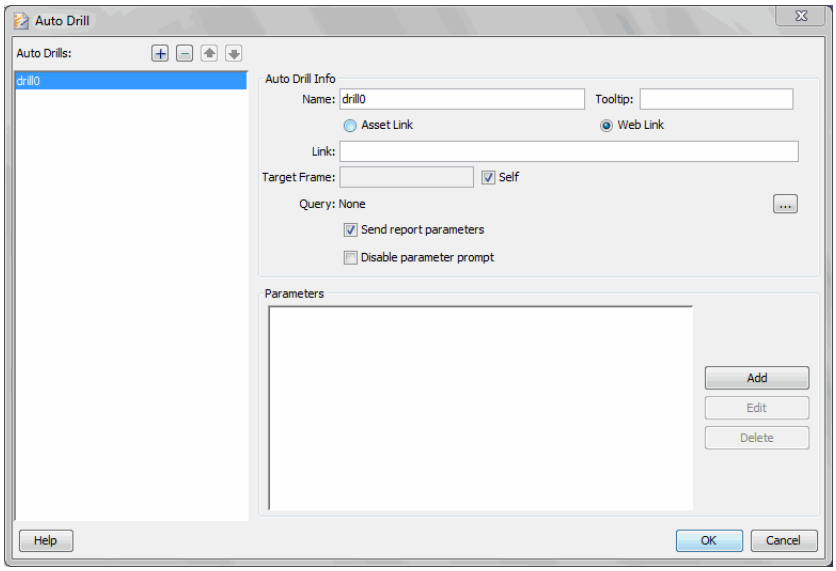
1. Open the **Asset** panel. Expand the 'Data Source' node and the 'Orders' data source.

2. Double-click on the 'Order Model' to open the model for editing.

3. Click on the cell in the 'Auto Drill' column corresponding to the 'Customer' > 'Company' attribute. This opens the 'Auto Drill' window for the 'Company' attribute.



4. In the ‘Auto Drill’ window, click the ‘New Drill’ button to create a new auto-drill scheme. The new drill is named ‘drill0’ by default. (If you add more than one drill to a given attribute, the end-user will be presented with a menu of hyperlink choices corresponding to the different drills you have specified.)



5. Rename the new drill by entering a name in the ‘Name’ field. (We use “DrillToOrderList” as the name of the drill.)

6. Enter a tooltip in the ‘Tool Tip’ field. (In the example we use “List orders for company” as the tooltip.) The tooltip is displayed when the end-user hovers the mouse over a hyperlink. If there are multiple drills for a given attribute, the tooltip of each drill is displayed when the end-user hovers the mouse over the name of the drill in the menu of hyperlink options.

7. Click the ‘Asset Link’ button. From the ‘Link’ popup menu, select the ‘OrderList’ report.



This 'Link' menu is populated from the *repository.xml* (reports) and *asset.dat* (Viewsheets) repository files, which contain the lists of deployed assets. You can also manually enter the name of a deployed asset in the 'Link' field.

If a target report is configured to display parameter sheets when launched, you can bypass these screens by selecting 'Disable parameter sheets' on the 'Auto Drill' screen.

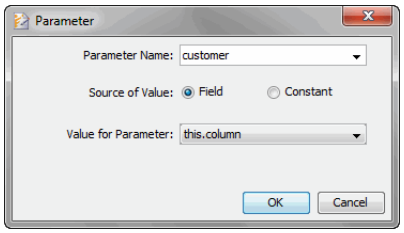
8. (Optional) If you want the target report to be displayed in a particular browser window, enter the name of the desired window in the 'Target Frame' field. (The specified name becomes the DOM 'name' attribute.) To specify that the target report should open in the User Portal frame, select the 'Self' checkbox.

9. Press the 'Add Parameter' (+) button to open the 'Parameter' dialog box.

10. From the 'Parameter Name' menu, select the '`customer`' report parameter. This is the report parameter that will receive the passed value.

    The parameter name that you specify in the 'Parameter Name' field must match the parameter name specified in the target report. (By selecting a name from the 'Parameter Name' menu, you ensure that the parameter name matches. You can also manually type a parameter name into the 'Parameter Name' field if the parameter has not yet been defined in the report.) See *Parameterization* in *Advanced Report Design* for more information about configuring report parameters.

11. From the 'Value for Parameter' menu, select '`this.column`'. The variable '`this.column`' represents the actual data values in the column.



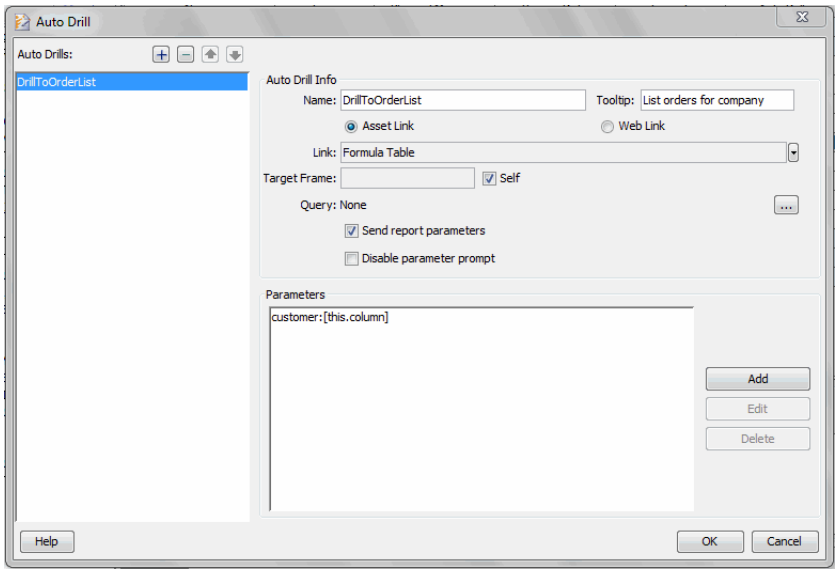
12. Press 'OK' to close the 'Parameter' dialog box.

Report parameters are parameters defined within the report containing the hyperlinks.

13. Check the 'Send Report Parameters' option to send report parameters together with the hyperlink (in addition to the auto-drill parameters).

14. Click the 'OK' button to complete the auto-drill configuration.

15. Save the changes to the data model by clicking the ‘Save’ button. The final configuration of the ‘Auto Drill’ window for this example is shown below.



To test the auto-drill you have added to the ‘Order Model’, you will now create a new report that uses the `Customer.Company` attribute.

16. Create a new ‘Blank Tabular’ report. (See *Creating a New Blank Report* in *Advanced Report Design* for more information.)

17. Click the ‘Table’ component in the **Toolbox** panel to start the ‘Data Binding’ wizard.

18. Under the **Data** tab of the ‘Data Binding dialog box, bind the table to the ‘Order Model’ data source.

19. In the **Columns** tab of the ‘Data Binding dialog box, select the following columns from ‘Order Model’: `Customer.Company`, `Customer.City`, `Customer.State`, and `Customer.Zip`. (`Customer.Company` is the attribute that you configured for auto-drill.)

20. Click ‘Finish’ to exit the ‘Data Binding’ dialog box, and deploy the report to the server. (See *Deploying an Asset* for more information.)

When the user views this report in the User Portal, they will see automatic hyperlinks on all the values in the ‘Company’ column. (The ‘State’ column also shows hyperlinks because this attribute was pre-configured for auto-drill.) When the user clicks on one of the ‘Company’ hyperlinks, this launches the ‘OrderList’ report in a new browser window, and the

'OrderList' report displays *only* the orders for the specific company that the user clicked.



All other reports and Viewsheets that use the `Customer.Company` attribute of the Order Model will now also automatically provide these same drilldown hyperlinks to the 'OrderList' report.

This example illustrated how to pass parameters in an auto-drill hyperlink to a report. You can also pass query-based parameters to a report. The procedure is the same as passing query-based parameters to a web page, which is described in *Example 2: Passing query-based values in drill*. To auto-drill to a report, select the 'Asset Link' option, and be sure to match the drill parameter names to the report parameter names.

Example 2: Passing query-based values in drill

In *Example 1: Passing column values in a drill*, you learned how to create an auto-drill hyperlink that passes a value that the user clicks as a parameter to a report. Sometimes, however, you do not need to transmit the selected value itself, but rather some information that is *related* to the selected value. To do this, you can add a query-based auto-drill to the logical model.

Walkthrough

In this example, you will add an auto-drill that transmits a product ID when the user clicks the corresponding product name in a report or Viewsheet. This example also illustrates how to target a *web page* with the hyperlinks (rather than a report, as in *Example 1: Passing column values in a drill*).

To add a query-based auto-drill, the first step is to build an appropriate query. In the following sequence, you will use the Query Wizard to build a new query that accesses the desired product information. You will design this new query to return the data you wish to pass as parameter in the auto-drilldown, which is the 'PRODUCT_ID' field.

```
The auto-drill query should accept parameters, and return rows
containing the appropriate values to pass to the target report or
web page.
```

1. Open the **Asset** panel. Expand the 'Data Source' node.

For more information on constructing queries, see the *Independent Query* section.

2. Right-click the 'Orders' data source, and select 'New Query' from the context menu. This opens the 'Query Wizard'.

3. On the **General** tab, enter a name for the new query in the 'Name' field (e.g., 'getProdID'). Click 'Next'.

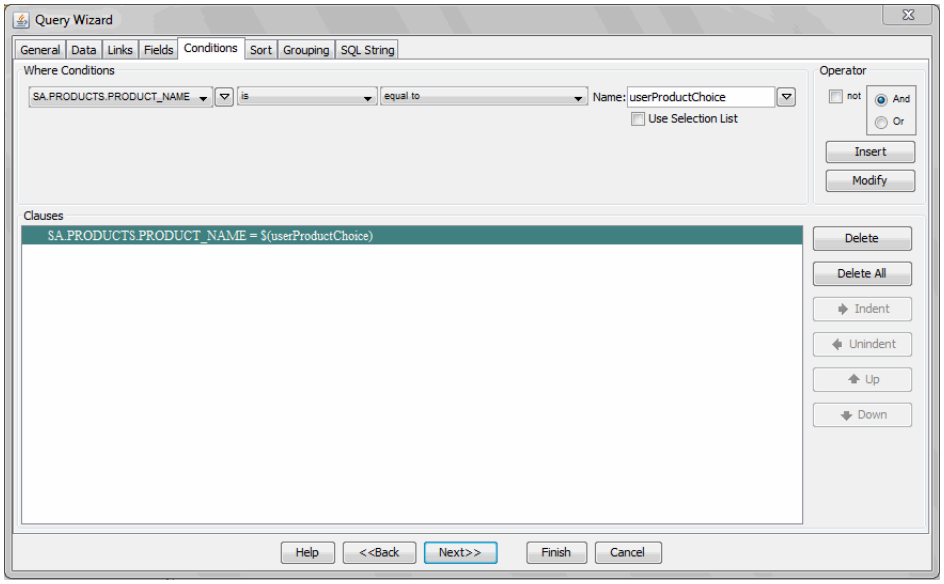
4. Under the **Data** tab of the Query Wizard, select the 'PRODUCTS' table from the 'TABLE' > 'SA' node.

5. Under the **Fields** tab of the Query Wizard, select the 'PRODUCT_ID' field.

6. Under the **Conditions** tab of the Query Wizard, select 'SA.PRODUCTS.PRODUCT_NAME' as the conditioned field, since this will be the input to the query (the attribute that the user will be clicking in the report/viewsheet).

7. Select 'is equal to' from the adjacent popup menus.

8. Select 'Variable' from the right-most popup menu, and enter a variable name in the text field. For this example, enter `$(userProductChoice)` as the variable name.



Note: If a query returns multiple rows, the user is shown a pop-up with all of the values. The user can then clarify their selection by choosing one of these to send to the report or URL.

9. Click the 'Append' button to add this condition.

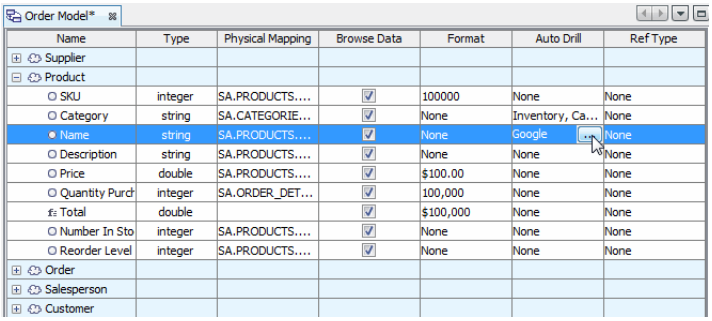
10. Click the 'Finish' button in the Query Wizard, and then click the 'Save' button in the toolbar to save this query.

In the next sequence of steps, you will use the saved query as the basis of a new auto-drill:

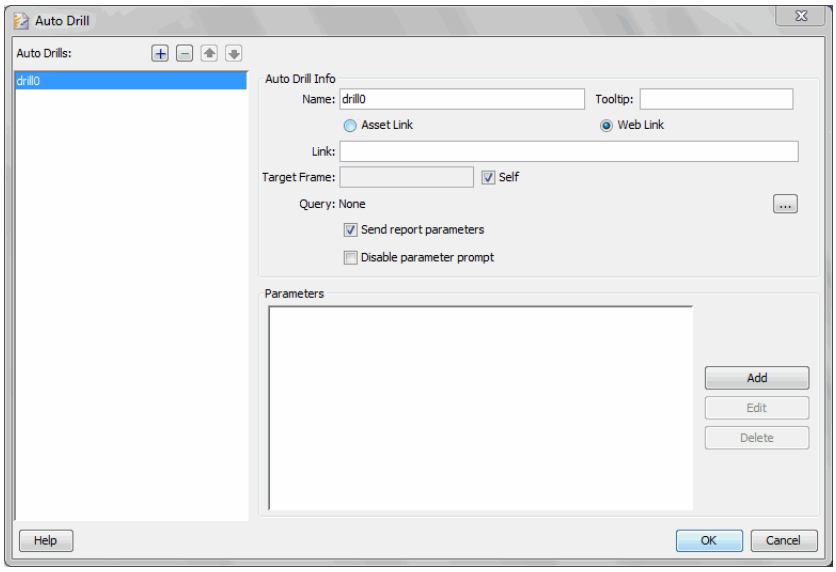
1. Open the **Asset** panel. Expand the ‘Data Source’ node and the ‘Orders’ data source.

2. Double-click on the ‘Order Model’ to open the model for editing.

3. Click on the cell in the ‘Auto Drill’ column corresponding to the ‘Product’ > ‘Name’ attribute. This opens the ‘Auto Drill’ window for the ‘Name’ attribute.



4. In the ‘Auto Drill’ window, click the ‘New Drill’ button to create a new auto-drill scheme. The new drill is named ‘`drill0`’ by default. (If you add more than one drill to a given attribute, the end-user will be presented with a menu of hyperlink choices corresponding to the different drills you have specified.)



5. Rename the new drill by entering a name in the ‘Name’ field (e.g., “SendProdID”).

6. Enter a tooltip in the ‘Tool Tip’ field (e.g., “View product information”). The tooltip will be displayed when the end-user

hovers the mouse over the hyperlink. If there are multiple drills for a given attribute, the tooltip of each drill will be displayed when the end-user hovers the mouse over the name of the drill in the menu of hyperlink options.

This URL is for illustrative purposes since the 'PRODUCT_ID' attribute is an integer value and will not yield any useful search results.

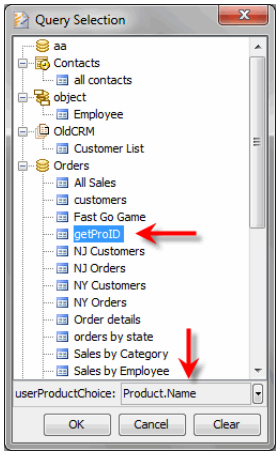
7. Select the 'Web Link' button. In the 'Link' field, enter the URL of the hyperlink destination with parameters omitted. For example, if the hyperlink is to pass a value as parameter to the Google search engine, the 'Link' field should specify "`http://www.google.com/search`".



8. (Optional) If you want the target web page to display in a particular browser window, enter the name of the desired window in the 'Target Frame' field. (The specified name becomes the DOM 'name' attribute.) To specify that the target page should open in the User Portal frame, select the 'Self' checkbox.



9. Click the 'Query' button in the 'Auto Drill' window to bring up the 'Query Selection' dialog box. Select the saved query ('getProdID'), choose 'Product.Name' for the 'userProductChoice' dropdown at the bottom, and click 'OK'.



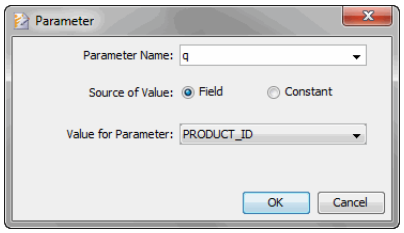
`Note: If the query takes multiple parameters, you can connect them to various attributes/columns from the current model/query. If those columns are present in the table that is displaying the auto-drill, the values of those columns will also be passed to the subquery.`

10. Press the 'Add Parameter' (+) button to open the 'Parameter' dialog box.

    Note that the 'Value of Parameter' menu displays the output of the 'getProdID' query, which is the attribute called 'PRODUCT_ID'.

The ‘PRODUCT_ID’ attribute is available only as a result of running the ‘getProdID’ query; it is not explicitly shown anywhere in the user’s report.

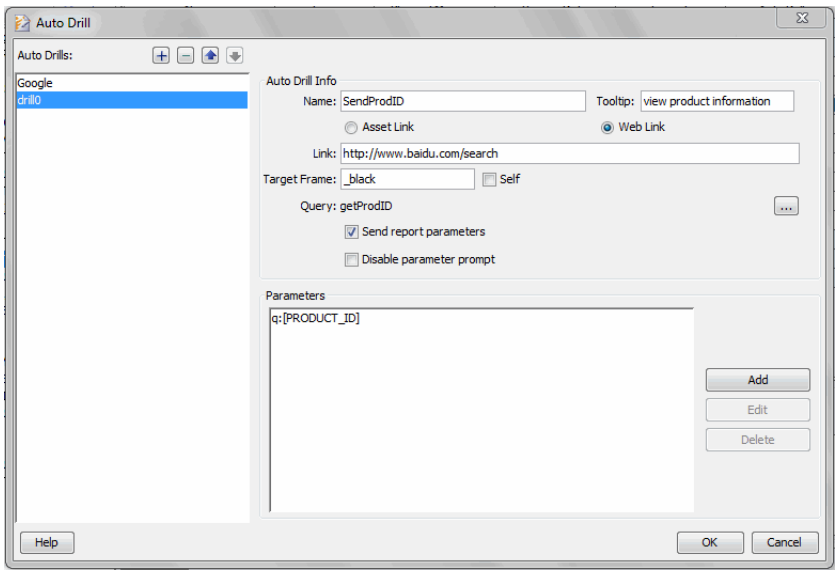
11. Enter the variable name “q” in the ‘Parameter Name’ field. This name will be the parameter name used in the URL that is automatically constructed. (The parameter name “q” is the parameter name expected by the Google search engine.)



12. Select ‘PRODUCT_ID’ in the ‘Value for Parameter’ menu. This indicates that ‘PRODUCT_ID’ will be passed as a parameter in the auto-drill.

13. Press ‘OK’ to close the ‘Parameter’ dialog box.

14. Press ‘OK’ to exit the auto-drill configuration. The final configuration of the ‘Auto Drill’ window for this example is shown below.



The hyperlinks constructed by this auto-drill will be of the following form:

```
http://www.google.com/search?q={ProductID}
```

where `{ProductID}` is the value of 'PRODUCT_ID' corresponding to the 'Product.Name' clicked by the user. All reports and Viewsheets that use the 'Product.Name' attribute of this logical model will now automatically provide hyperlinks of the above form on that attribute.

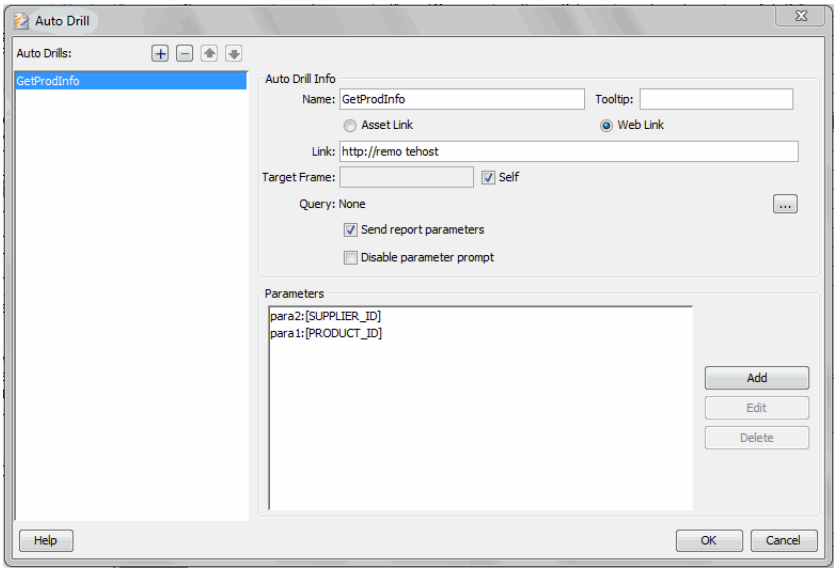
For an illustration of how these hyperlinks will appear in a deployed report, see Steps 16–20 of *Example 1: Passing column values in a drill*.

Passing Multiple Parameters in a Query-Based Auto-Drill

The prior examples (*Example 1: Passing column values in a drill* and *Example 2: Passing query-based values in drill*) described how to use auto-drilldowns to pass a single parameter to a specified report or web page. It is also possible to pass multiple parameters in an auto-drill.

The procedure for creating multiple-parameter drills is the same as described in the previous sections. If the query specified in Steps 1–10 of *Example 2: Passing query-based values in drill* returns more than one column, all of the returned columns will be available for selection in the 'Value' pane of the 'Auto Drill' page (Steps 9–10).

For example, if a query called 'getProductInfo' accepts a parameter and returns the two columns 'PRODUCT_ID', and 'SUPPLIER_ID,' then when you specify the 'getProductInfo' query in the 'Auto Drill' window, these two columns will be shown in the 'Value' pane. Each of the multiple attributes listed in the 'Value' panel can then be selected and mapped to a parameter name.



As in the previous examples, if the auto-drill is targeting a report, the parameter names specified in the 'Auto Drill' window must match the

corresponding parameter names specified in the report. If the auto-drill is targeting a web page, the parameter names specified in the 'Auto Drill' window will be the names used in the hyperlink URL, with parameter-value pairs separated by the ampersand (&) symbol. For the 'Auto Drill' settings shown above, the resulting hyperlinks will have the form

```
http://remotehost?param1={value1}&param2={value2}
```

where `param1`, `param2` are the parameter names specified in the 'Auto Drill' window, and `{value1}`, `{value2}` are the values returned by the query for the user's particular hyperlink click.

If an auto-drill query returns multiple rows, the user is prompted to select one of the possible parameter combinations. For example, consider a case where the user clicks on a 'State' field to see a list of orders for a customer. If there is only one customer in a given State, the drill-down goes directly to the detail report. However, if there are multiple customers in the selected state, the user is first prompted with a list of these customers. After they submit their selection, the detail report returns data for the chosen customer.

Adding Auto-Drilldown to a Query

Adding auto-drilldown at the query level is very similar to adding auto-drilldown at the model level. (See *Example 1: Passing column values in a drill* and *Example 2: Passing query-based values in drill* above.) You can add auto-drills to new queries or existing queries.

Walkthrough

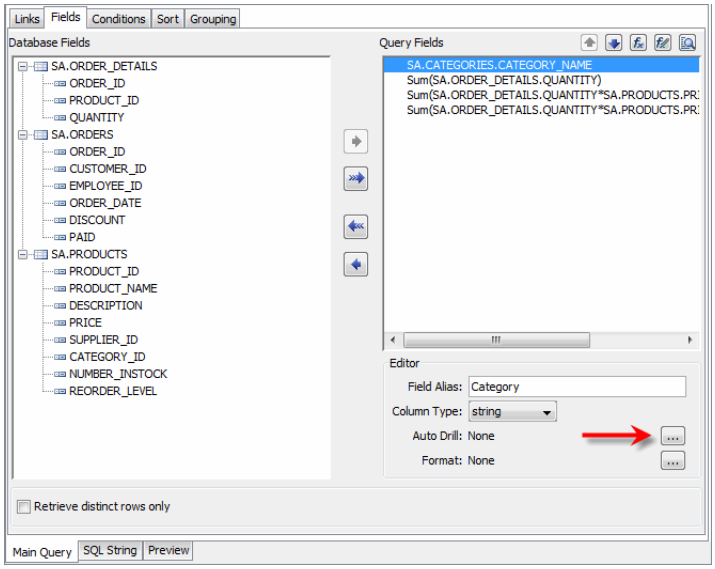
In the following example, you will add an auto-drill to the 'Sales by Category' query of the 'Orders' data source:

1. Open the **Asset** panel. Expand the 'Data Source' node and the 'Orders' data source.

2. Double-click the 'Sales by Category' query under the 'Orders' data source. This opens the query for editing.

3. Select the **Fields** tab at the top of the query panel. (Make sure that the 'Main Query' tab at the bottom of the panel is selected.)

4. Select '`SA.CATEGORIES.CATEGORY_NAME`' in the 'Query Fields' panel. This enables the 'Editor' at the bottom of the panel.





5. In the editor, click the ‘Auto-drill’ button.

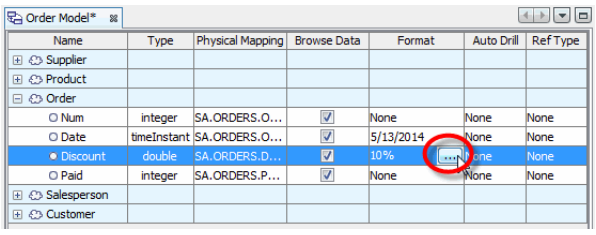
   The ‘Auto-Drill’ window that now opens is the same as the ‘Auto-Drill’ window for data models. For further instruction on using this ‘Auto-Drill’ window, see *Example 1: Passing column values in a drill* and *Example 2: Passing query-based values in drill*. After you add the auto-drill to this query field, all reports and viewsheets that use the ‘Sales by Category’ query will automatically provide hyperlinks.

Note that the ‘Auto-Drill’ window for query-level auto-drill also offers the option of passing query-based parameters. See *Example 2: Passing query-based values in drill* for details of the procedure.

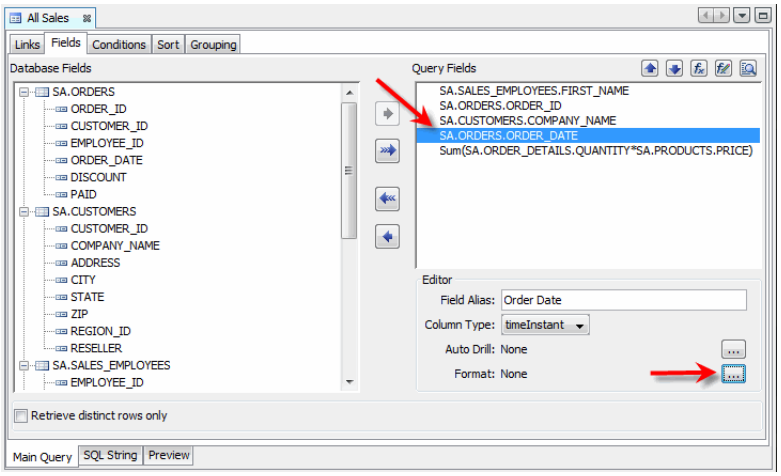
3.4.3 Adding a Format to a Data Field

To associate a data format with an attribute (column) at the model level or query level, follow the steps below:

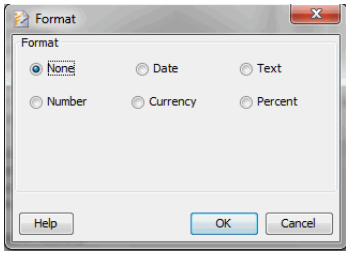
1. For a *data model*, open the model for editing, and click the button in the ‘Format’ column corresponding to the desired attribute.



For a *query*, open the query for editing, and select the desired field in the 'Query Fields' panel. Then click the 'Format' button in the 'Editor' panel.



This opens the 'Format' dialog box shown below.



2. In the 'Format' dialog box, select the desired formatting option. The options are the same as those available in the **Format** tab of the Style Studio. (See *Formatting* in *Advanced Report Design*.)

For date and number formats, you can specify an optional format pattern string. The date format pattern must conform to the patterns specified by the `java.text.SimpleDateFormat` class. For example, to format a date as 'Jan 01, 2000', use '`MMM dd, yyyy`'.

Decimal formatting only applies if the value is a numeric type.

The number format pattern must conform to the patterns specified by the `java.text.DecimalFormat` class. For example, to add commas to a number, use '`#, ###, ###. ##`'. A '`#`' will show a digit if needed, and a '`0`' will display zero if the digit is not needed. Adding a '`%`' at the end of the format will multiply the value by 100 and display as a percentage.

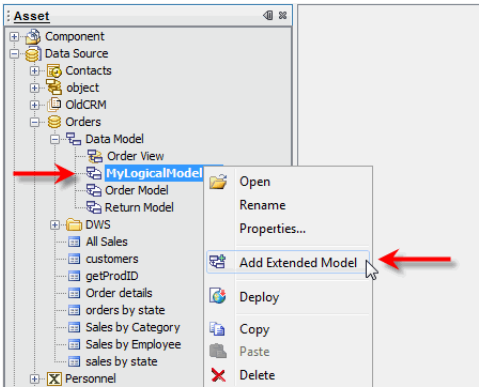
3.4.4 Extending a Logical Model

In a multi-tenant environment, you can provide a different logical model to each tenant. Each logical model can provide access to a tenant-specific physical view that draws data from a tenant-specific database connection.

To create a tenant-specific logical model, *extend* an existing logical model by linking the model to an existing extended physical view. Follow the steps below:

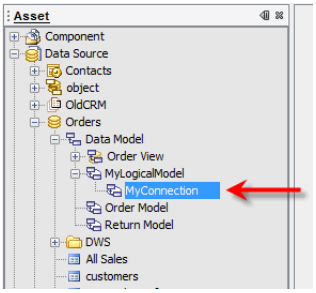
1. Open the **Asset** panel, and expand the 'Data Source' node.

2. Right-click on the base logical model that you wish to extend, and select 'Add Extended Model' from the context menu. This opens the 'Extended Model' dialog box.

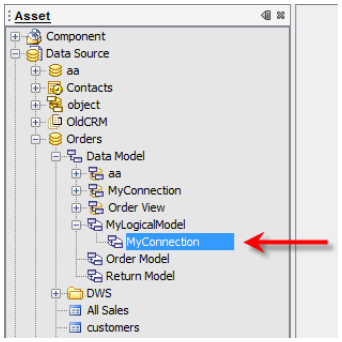


Note: An extended model is associated with one (and only one) connection and physical view. Likewise, a connection can be associated with only one extended model.

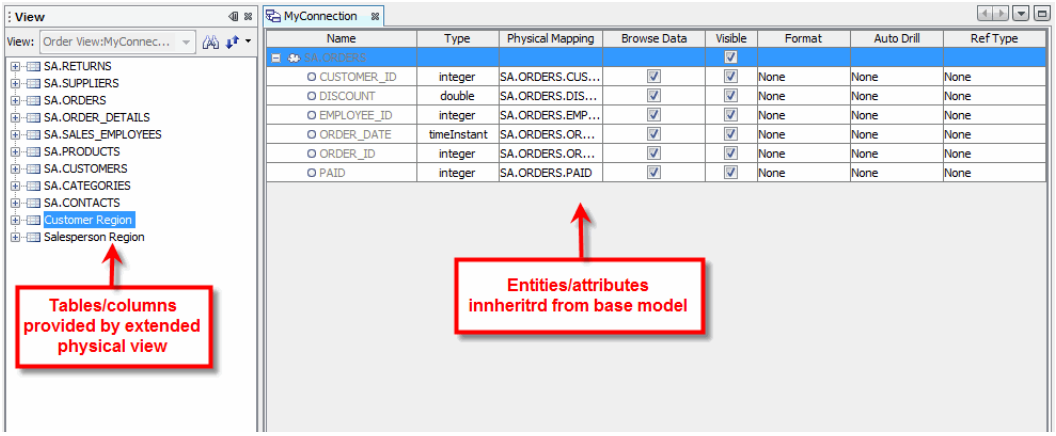
3. In the 'Extended Model' dialog box, choose a connection option for the new extended model, and click 'OK'. (You can select the default connection for the data source, or any of the additional connections.)



This adds the new extended model under the 'Data Source' node, and opens the extended model for editing.



The **View** panel displays the tables and columns that are available from the corresponding extended physical view. The right panel displays entities and attributes that are inherited from the base model.



Note that entities and attributes which are inherited from the base logical model cannot be modified within the extended model. However, you can hide an inherited property by disabling its 'Visible' option. (See *Modifying Entities and Attributes* for more information.)

4. Drag tables or columns from the **View** panel into the model panel to add new entities and attributes, or to extend inherited entities. (See *Creating a Logical Model* for more information about creating entities and attributes.)

5. When you have finished defining the extended model, click the 'Save' button in the Style Studio toolbar.

You can now expose this extended model to particular tenants by setting user or group permissions on the corresponding *connection*. See *Component/Object Permissions* in the *Advanced Server Administration* for more information on authorizing connection access.

See Also

Adding Additional Connections, for information on configuring connections.

Extending a Physical View, for information on adding an extended view.

Creating a Logical Model, for information on creating the base logical model.

3.5 Independent Query

For cases when you need greater control over data access or more sophisticated logic than that provided by a generic data model, you can create an independent *query*.

A query is a specific request for data from the database, and often incorporates advanced or proprietary database features. A query is similar to a relational database *view*, except that a query is normally not generic enough to be created as permanent database object. Nevertheless, Style Intelligence queries can be quite flexible, depending on the query parameters and data selection.

For non-relational data sources, queries are often the *only* available means to obtain the required dataset. In this situation, a generic query can indeed serve the same role as a database view. For relational databases, Style Intelligence also treats stored procedures and other query derivatives as queries.

You can create a query by using simple drag-and-drop operations, or by manually entering a valid SQL string. (See *Creating a New Query* and *Editing a Query SQL String*, respectively.) You can choose to save the query publicly, so that it will available to all Viewsheets, Worksheets, and reports, or you can choose to embed the query within a specific report.

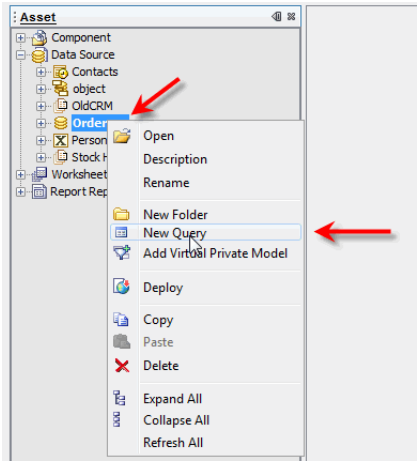
Queries and data models are independent of one another. However, you can transform and combine queries with data models for use in more sophisticated applications by using the Data Block technology provided by Data Worksheets. (See *Advanced Data Mashup* for more details.)

3.5.1 Creating a New Query

To create a new query for a relational database, follow the steps below:

1. Open the Style Studio **Asset** panel, and expand the ‘Data Source’ node.

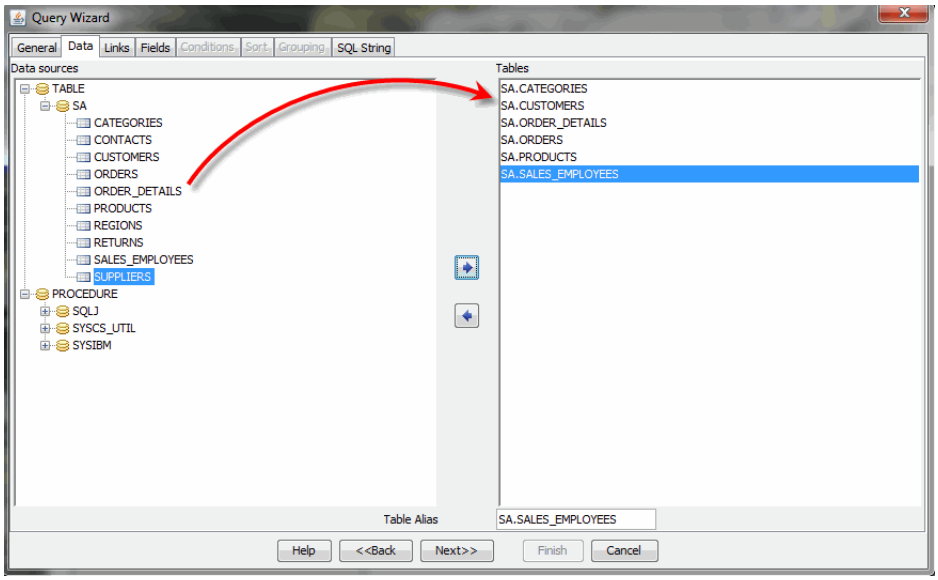
2. Right-click on the data source that you wish to query, and select ‘New Query’ from the context menu. This opens the ‘Query Wizard’ dialog box.



3. Under the **General** tab of the 'Query Wizard' dialog box, enter a name for the query and (optional) description. Click 'Next' to advance to the next tab.

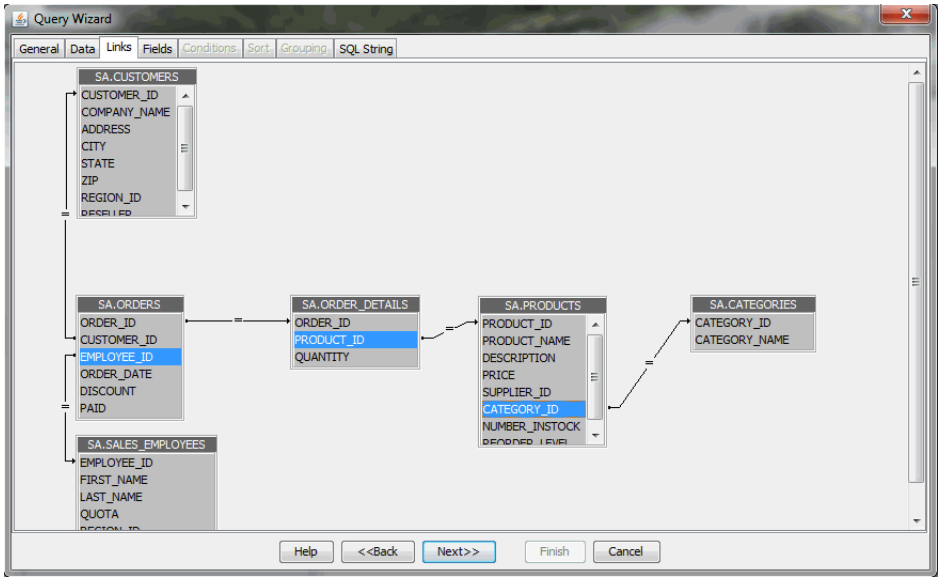
You can also double-click fields to add them, or use the arrow buttons.

4. Under the **Data** tab, drag the desired database tables from the 'Data Sources' panel to the 'Tables' panel. Click 'Next' to advance to the next tab.



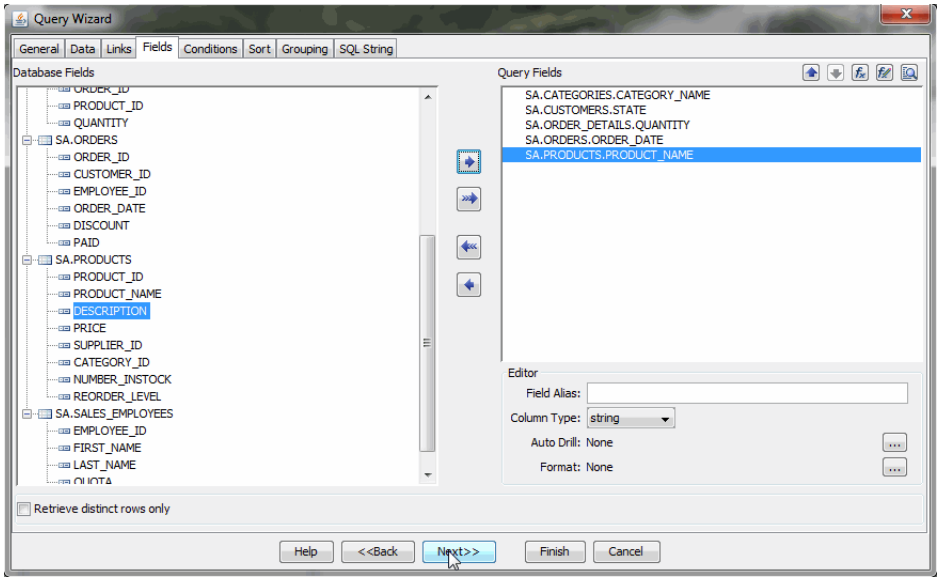
To manually enter SQL rather than using the Wizard, click the **SQL String** tab. See *Editing a Query SQL String*.

5. Under the **Links** tab, define a set of joins to relate the tables. This process is the same as defining joins for physical views. (See *Defining Join Relationships* for an explanation of how to relate tables.) Click 'Next' to advance to the next tab.



You can also double-click fields to add them, or use the arrow buttons.

6. On the **Fields** tab, drag the fields that you wish the query to return from the 'Database Fields' panel to the 'Query Fields' panel.



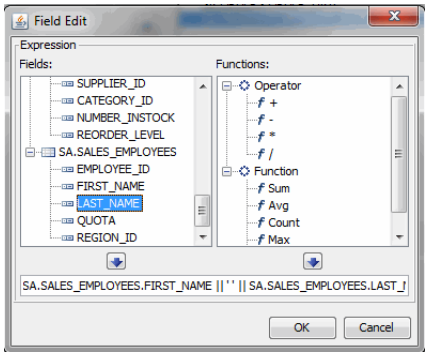
7. (Optional) To create a derived (expression) column, follow the steps below:



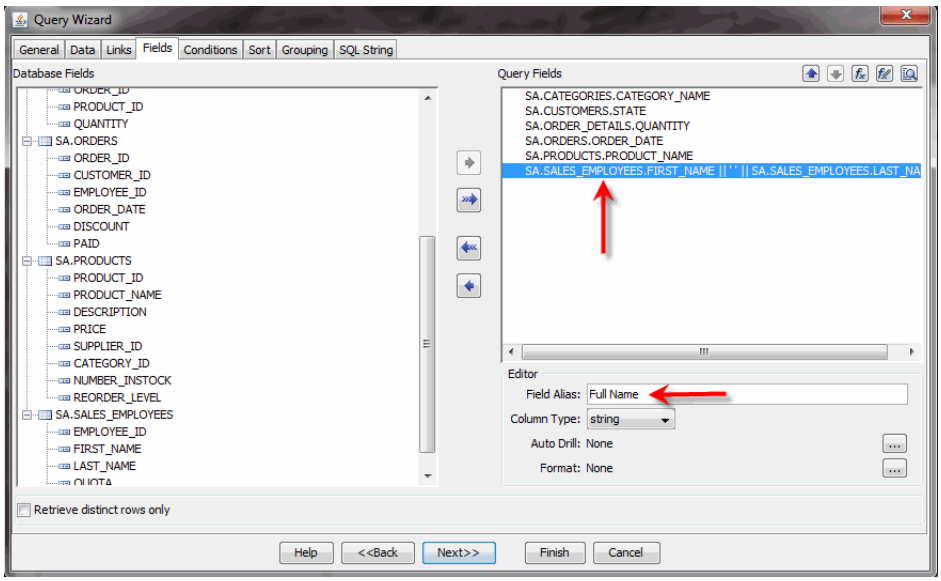
a. Click the 'Add Expression' button above the 'Query Fields' panel. This opens the 'Field Edit' dialog box.

Double-click the field names to add them to the expression.

b. Enter an expression to define the new column, then click 'OK'. (The expression should use the appropriate SQL syntax for the underlying database.)

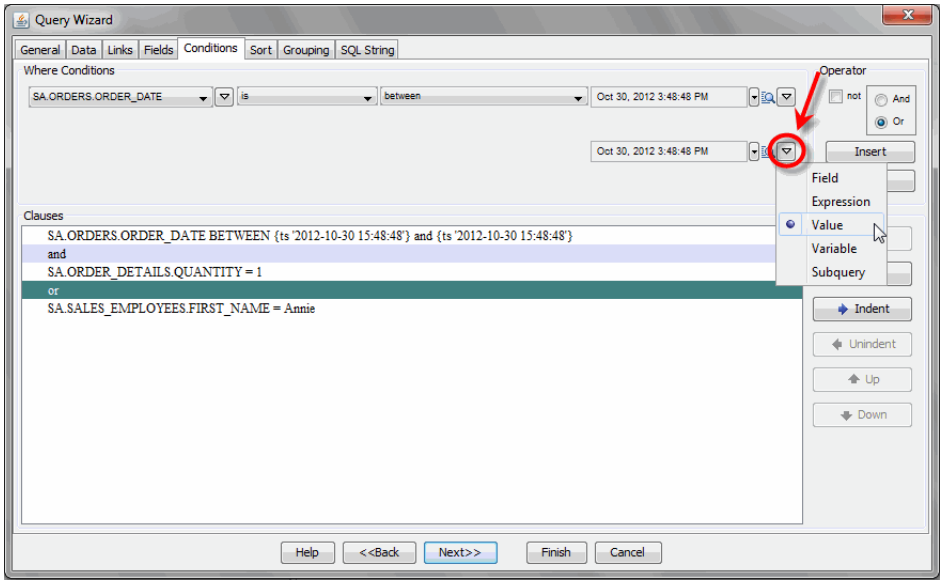


c. To create an alias for the expression column, select the expression column in the 'Query Fields' panel, and type a name into the 'Field Alias' box in the bottom panel.



d. Click 'Next' to advance to the next tab.

8. On the **Conditions** tab, use the menus to specify a filtering condition. This condition will restrict the data returned by the query. Click the "arrow" button on the right side to select a method of supplying the right-hand side of the condition.



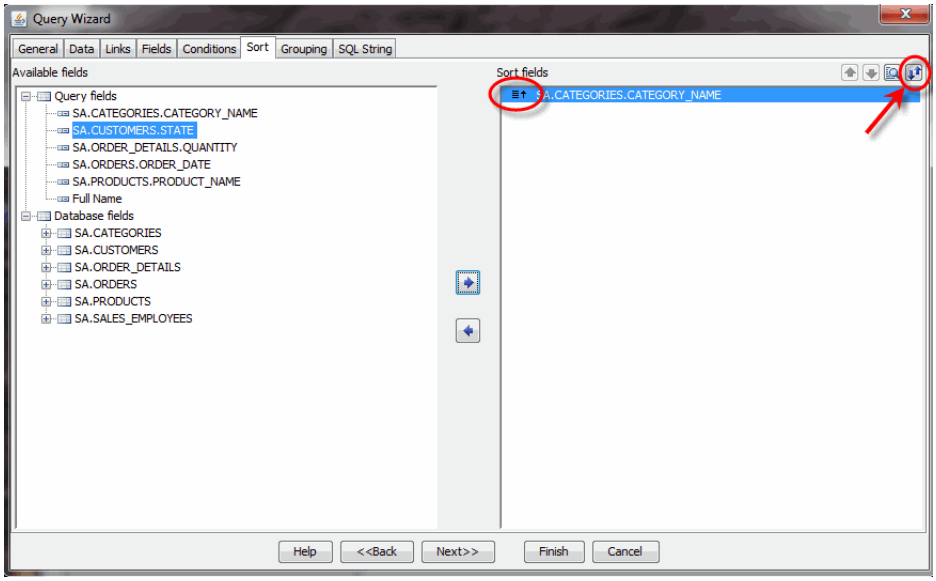
When you enter a fixed value as the right-side of the condition, use the 'Browse Data' button to assist the selection.

The 'Field' option allows you to select a different column from the same result set. 'Expression' allows you to enter a regular expression using Perl5 regex syntax (see A.5 *Regular Expressions*.) 'Value' allows you to enter a fixed value. 'Subquery' allows you to specify a distinct query to return the right-side of the condition. 'Variable' allows you to enter an arbitrary variable name. The value of the variable will either be provided by the user, by script, or by another subquery. Check the 'Selection List' box to prompt the user with a list of values read from the field. (See *Query Properties* for more details about variables.)

9. Click 'Next' to proceed to the **Sort** tab. Drag the fields on which you wish to sort from the 'Available Fields' panel to the 'Sort Fields' panel.

An arrow to the left of the field name indicates the current sort order.

10. Select a field in the 'Sort Fields' panel, and click the 'Order' button above the panel. Click once to sort in descending order. Click again to sort in ascending order.



The table below summarizes the operation of the different tabs in the 'Query Wizard' dialog box. The same tabs are available also when editing a query. See *Editing a Query* for more details.

Table 2. Query Definition Tabs

TAB NAME	DESCRIPTION
Links	Shows all tables selected in the query and their relationships (joins). Joins are created automatically if the data model contains join information. Right-click a join line to change the join properties.
Fields	Column selection and column alias definition. Data formatting and auto-drilldown options are the same as for logical models. See *Adding a Format to a Data Field* and *Auto-Drilldown* in the *Logical Model* section for more information. The 'Retrieve distinct rows only' option prevents the query from returning duplicate rows.
Conditions	Specifies query selection conditions.
Sort	Sorting order of the table.
Grouping	Grouping columns and 'Having' conditions. The having conditions can only be defined if grouping is defined.
SQL String	The resulting SQL string or directly-entered SQL statement. The 'Parse SQL' checkbox controls whether the statement is parsed by Style Studio into a structured query. For very complicated SQL statements that are slow to parse, you may wish to disable parsing. In this case, you should click 'Get Column Info' to get the meta-data by executing the query. Click it again whenever the meta-data (columns or data-types) change.

3.5.2 Editing a Query

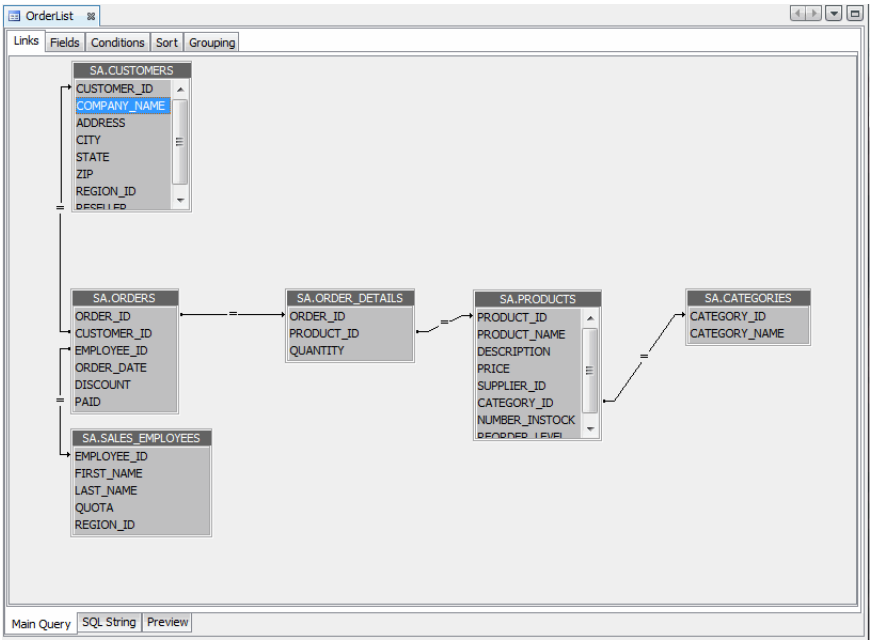
To edit an existing query, follow the steps below:

1. Open the **Asset** panel, and expand the ‘Data Source’ node.

2. Expand the data source that contains the query you want to modify.

```
If you make a change to the structure of the database schema, press
the ‘Refresh Metadata’ button on the toolbar to update the query.
```

3. Double-click the query. This opens the query for editing.



The query editor is organized using the same tabs as the ‘New Query’ dialog box: **Links**, **Fields**, **Condition**, **Sort**, and **Grouping** at the top, and **SQL String** at the bottom. See *Creating a New Query* and *Editing a Query SQL String* for instructions on using these tabs.

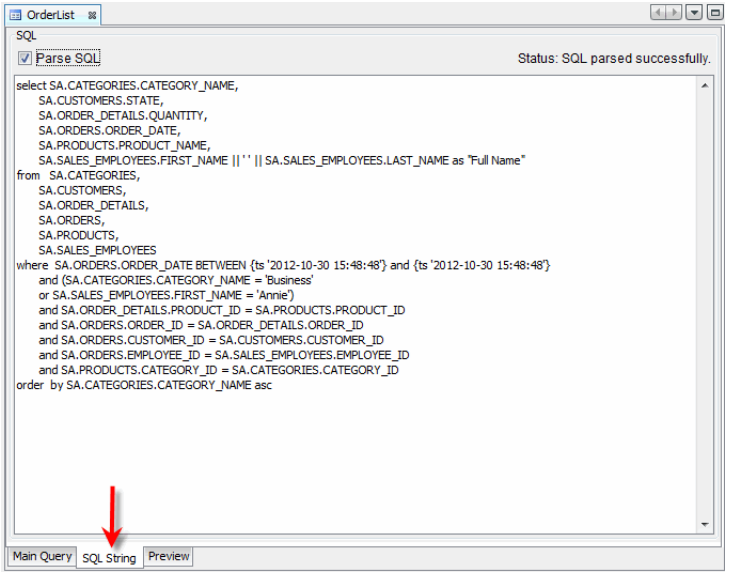
To view the results returned by the query, click the **Preview** tab at the bottom of the editor.

3.5.3 Editing a Query SQL String

To manually edit the SQL string for a query, follow the steps below:

1. Open the query for editing, as described in *Editing a Query*.

2. Click the **SQL String** tab at the bottom of the editor. This opens the SQL string for editing.



To change back to the graphical query view, click the **Main Query** tab.

3. (Optional) Select the ‘Parse SQL’ checkbox to allow Style Studio to automatically parse the SQL string and reconstruct the graphical query view. This is the default.

   Note, however, that due to database variations, the parser may not be able to parse all valid SQL statements. In such cases, you will be asked whether you want to keep the change or modify the query. In either case, manually entered SQL always overrides the query graphical query definition. (This allows any query to be constructed in Style Studio.) For very complicated SQL statements that are slow to parse, you may wish to disable the ‘Parse SQL’ option.

4. Click the ‘Save’ button in the Style Studio toolbar to save your changes.

You can add query variables to the SQL string using the `$(var)` construct, where *var* is an arbitrary variable name. This variable will be translated into a SQL variable and the value will be provided at runtime.

You can also specify a special string replacement variable with the `$(@var)` construct. This allows you to dynamically construct the SQL string, through scripting, in order to satisfy complex business requirements. For example, you can dynamically determine the table from which to select at runtime. To do this, define a partial SQL string, such as “`select id from $(@var)`”. Then create a runtime script to replace the variable `var` with a table name such as ‘`customers`’ or ‘`prospects`’.

3.5.4 Creating a Local Query

A *local query* is a query that is embedded within a report, rather than being stored in the common *query.xml* repository file. Embedding a query data source within a report enhances report portability. However, because a local query is available only to the particular report in which it is embedded, embedding is most useful in cases when the query is unique to the report and will not be reused for other reports.

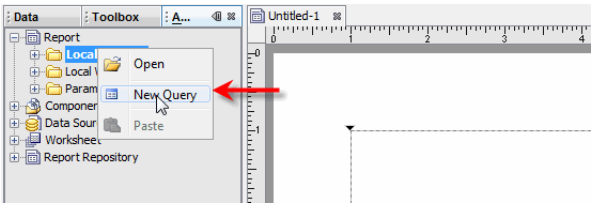
Creating a Local Query

To create a local query, you must first open a report. Follow these steps:

1. In Style Studio, open the report into which you want to embed the local query.

2. Expand the Style Studio **Asset** panel.

3. Expand the 'Report' node, and right-click the 'Local Query' folder. Select 'New Query' from the context menu.




```
Note: The 'Local Query' folder is only available when a report is
open for editing in Style Studio.
```

4. Design the query as desired, and save the query. (See *Creating a New Query* for full information.)

The query is displayed under the 'Local Query' folder, and is now embedded within the current report. You can bind this local query to a report element in the same way that you bind any other data source. See *Walkthrough: Choosing the Binding Data Source* in *Advanced Report Design* for more information.

See Also

Creating a Local Worksheet, in *Advanced Data Mashup*, for information on embedding a Worksheet.

Exporting a Local Query to the Global Repository

To export one or more local queries to the common query registry (*query.xml* file) so that they can be used by other reports, follow the steps below:

1. Open the report that contains the local query or queries you want to export.

2. Expand the **Asset** panel.

3. Expand the ‘Local Query’ node in the tree.

Alternatively, drag-and-drop the query to the desired location while holding down the **Ctrl** key.

4. To *copy* the query or queries from the ‘Local Query’ folder to the global repository (leaving the originals embedded in the report), follow these steps:

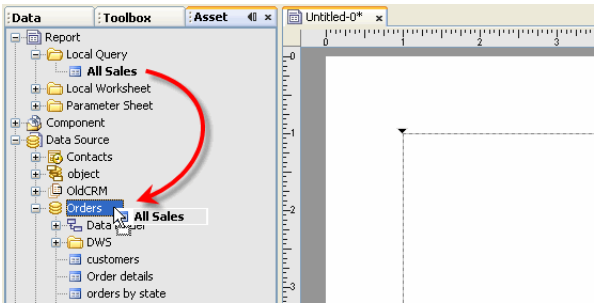
   a. Select the desired queries in the ‘Local Query’ folder, right-click, and choose ‘Copy’ from the context menu.

   b. Right-click on the desired folder under the ‘Data Source’ tree node, and select ‘Paste’ from the context menu.

5. To *move* the query or queries from the ‘Local Query’ folder to the global repository (removing them from their embedding in the report), follow these steps:

   a. Select the desired queries in the ‘Local Query’ folder.

   b. Drag-and-drop the selected queries on the desired folder under the ‘Data Source’ tree node.



The query or queries are added to the common repository. Elements in the report that were previously bound to removed local queries are automatically re-bound to the newly-exported global queries.

Importing a Local Query from the Global Repository

To import a query from the global query registry (*query.xml* file) into a local query, follow the steps below:

1. Open the report into which you want to import a query or queries.

2. Expand the Style Studio **Asset** panel.

3. Expand the ‘Report’ node, which contains the ‘Local Query’ folder.

4. Expand the data source that contains the query or queries that you wish to import.

Alternatively, drag-and-drop the query to the desired location while holding down the **Ctrl** key.

5. To *copy* the queries to the ‘Local Query’ folder (leaving the originals in the global repository), follow these steps:

   a. Select the desired queries, right-click, and select ‘Copy’ from the context menu.

   b. Right-click on the ‘Local Query’ folder, and select ‘Paste’ from the context menu.

6. To *move* the query or queries to the ‘Local Query’ folder (and remove them from the global repository), follow these steps:

   a. Select the desired queries.

   b. Drag-and-drop the selected queries to the ‘Local Query’ folder.

The query or queries are now displayed under the ‘Local Query’ folder and embedded within the current report. You can bind a local query to a report element in the same way that you bind any other data source. See *Walkthrough: Choosing the Binding Data Source* in *Advanced Report Design* for more information

Report elements that were bound to removed global queries are automatically re-bound to the newly-created local copies.

3.5.5 Accessing a Stored Procedure

Stored procedures are compiled and stored in a database. There are a few major differences between a stored procedure and a SQL query:

- A stored procedure is invoked as a function call instead of a SQL query.
- Stored procedures can have parameters for both passing values into the procedure and returning values from the call.
- Results can be returned as a result set, or as an OUT parameter cursor.



Stored procedures are listed on the same tree as tables and views. Only one stored procedure may be selected per query. The stored procedure parameters are listed on the middle pane. Specify values for the parameters by selecting each parameter on the tree and entering a value or variable name. If the result column list is not populated, select the ‘Column Info’ button to retrieve it. Select the ‘Preview’ button to preview the query. If any parameter is left as null or specified as ‘Prompt User’, when the query is

executed a parameter dialog will pop up to prompt for the remaining parameters.

Due to the different ways database vendors implement stored procedures, not all stored procedures are supported by JDBC. If a stored procedure cannot be used directly in the Style Studio, you can instead execute it with custom code, and set up as a Java Object data source.

The following is a list of known restrictions on stored procedures:

- Only one dataset is allowed in a stored procedure.
- In Oracle, only one cursor is allowed and it has to be the last parameter in the stored procedure.
- In Oracle, only standard SQL types should be used in the parameters (user defined types are not allowed).
- In PostgreSQL, create a free-form SQL query of the following form:

```
select * from procedureName(parameters)
```

- In MySQL, older drivers do not return procedure meta-data.

3.6 Virtual Private Model Security

A Virtual Private Model (VPM) is an optional component that provides a highly flexible and comprehensive security layer for ODBC and JDBC data sources. The VPM controls security for individual database tables. All queries, whether they are created manually or automatically generated from a data model, are filtered through the VPM.

Note: A VPM cannot be applied to *pre-generated* reports. See *Report Pregeneration* in *Advanced Server Administration* for more details.

The VPM is invisible to users, transparently providing access only to data for which a given user is authorized. Because the VPM is applied at the data level, security logic defined in the VPM applies to every method of data access (deployed report, ad hoc report, dashboard, Worksheet, etc.).

A VPM consists of *conditions*, *hidden columns*, *lookups*, and their associated triggers. Conditions and hidden columns allow you to filter rows and columns, respectively, and can be easily created for global application. Triggers provide fine-grained control on conditions and hidden columns by role, group, and even individual user. Multiple VPMs can be specified for a given data source, and these are joined in an ‘and’ fashion.

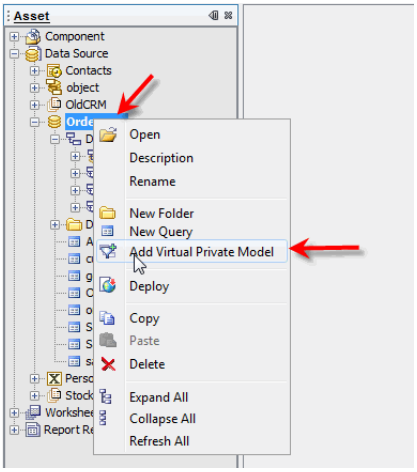
See Also
Design for Multi-Tenant Environment, in *Advanced Server Integration*, for connection-based access control.

3.6.1 Creating a VPM

To create a Virtual Private Model, follow these steps:

1. Open the **Asset** panel, and expand the ‘Data Source’ node.

2. Right-click the data source to which the VPM should be applied, and select ‘Add Virtual Private Model’ from the context menu.



This opens the Virtual Private Model Wizard, which contains several tabs for configuring the VPM. These allow you to set permissions on table rows (VPM conditions) and table columns (VPM hidden columns), and to set triggers to implement logic in permission settings. The following sections discuss this in greater detail.

3.6.2 VPM Conditions

You can add a condition to a VPM to filter the rows retrieved from the database tables. To add a condition to an existing VPM, follow these steps:

1. Open the **Asset** panel, and expand the ‘Data Source’ node.

2. Expand the data source that contains the VPM you want to edit. Double-click the VPM to open it for editing.

3. In the VPM edit view, click the **Conditions** tab at the bottom of the window.

4. In the ‘Define Conditions’ pane, click the ‘New’ button. This opens the ‘New Condition’ dialog box.

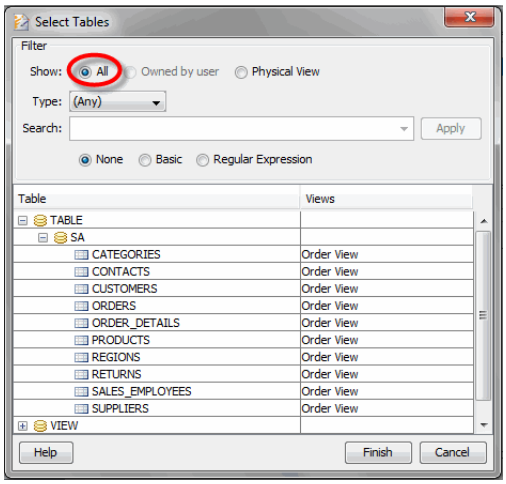
5. Enter a name for the new condition in the ‘New Condition’ dialog box, and click ‘OK’.

Previously defined conditions are listed in alphabetical order. (case insensitive).

6. In the ‘Define Conditions’ pane, click the ‘Choose Table’ button. This opens the ‘Select Table’ dialog box, showing a list of database tables.

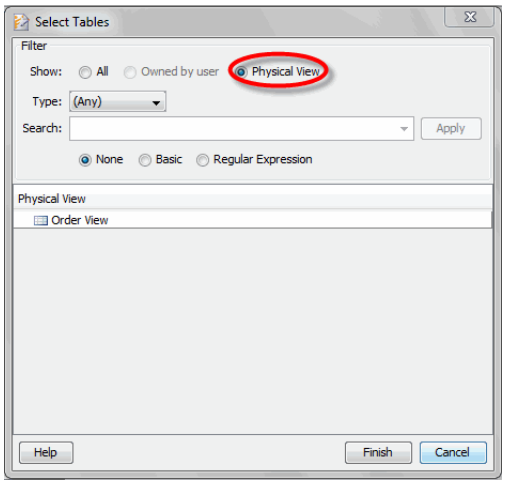
The dialog box provides controls for searching and sorting tables. See *Adding Database Tables to the Physical View* for details.

7. To define the condition on an individual table, select the ‘All’ option at the top. Select the desired database table from the list, and click ‘Finish’ to close the dialog box.



The filter condition can be based on any fields in the selected table, and the filter will be applied whenever the selected table appears in a query.

8. To define the condition on a physical view, select the ‘Physical View’ option at the top. Select the desired physical view from the list, and click ‘Finish’ to close the dialog box.



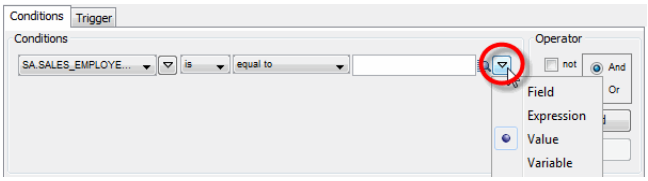
Note: If you have created one or more aliases of a table in the physical view, do not apply VPM conditions to the original table. (VPM conditions applied to the original table may affect its

aliases in an unpredictable manner.) Rather, apply VPM conditions only to the aliased copies.

For the ‘Physical View’ option, the filter condition can be based on fields from any table in the physical view. These fields will be automatically included in *any* query generated from the physical view, even if the query does not explicitly include them. This effectively applies the filter globally to all data models derived from the physical view.

9. In the ‘Conditions’ panel of the **Conditions** tab, specify the condition that you want to apply to the table.

   The menu controls for specifying the condition are the same as those used in query construction and data binding. See *Filtering Data in Data Binding Wizard* in *Advanced Report Design* for an example of constructing a condition. The arrow menu allows you to use different kinds of values in the right-side of the condition.



You can also use two special variables, _USER_ and _ROLES_, as parameters in the VPM condition to represent the current user user’s roles.

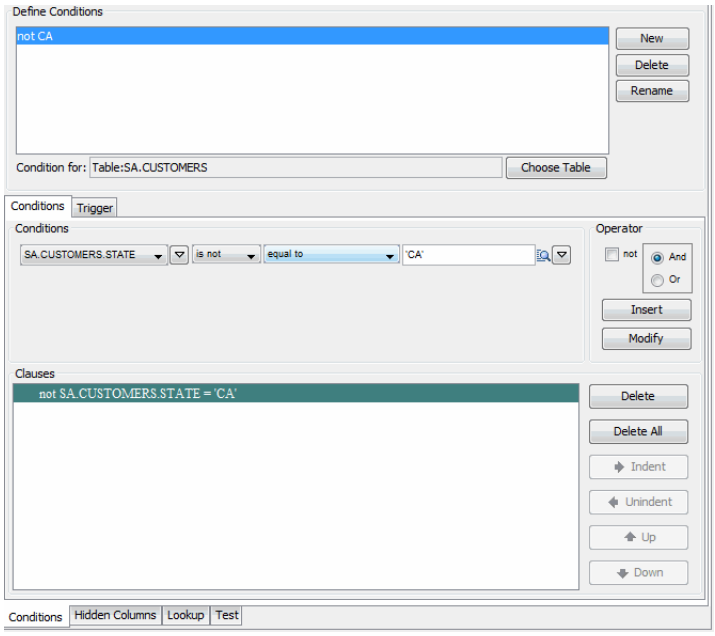
   ‘Field’ allows you to select a different column from the same table or physical view. ‘Expression’ allows you to enter a regular expression using Perl5 regex syntax (see A.5 *Regular Expressions*.) ‘Value’ allows you to enter a fixed value. ‘Variable’ allows you to enter the name of a parameter to supply the value at runtime. ‘Subquery’ allows you to specify a distinct query to return the right-side of the condition. ‘Session data’ provides the logged-in user’s name, an array of user roles, and an array of user groups.

10. Click ‘Append’ to add the condition clause. Repeat the above steps to add additional clauses, as needed.

11. Click ‘Save’ in the Style Studio toolbar to save the changes to the VPM.

Once a condition is added to the table or physical view, the selected table or physical view is automatically listed in the **Lookup** tab. This indicates that the VPM will be applied to any query that contains the selected table, or any data model that uses the physical view.

Figure 1. VPM Conditions



3.6.3 VPM Hidden Columns

You can specify columns of a table to which VPM access control should be applied. To add access control to table columns, follow these steps:

1. Open the **Asset** panel, and expand the ‘Data Source’ node.

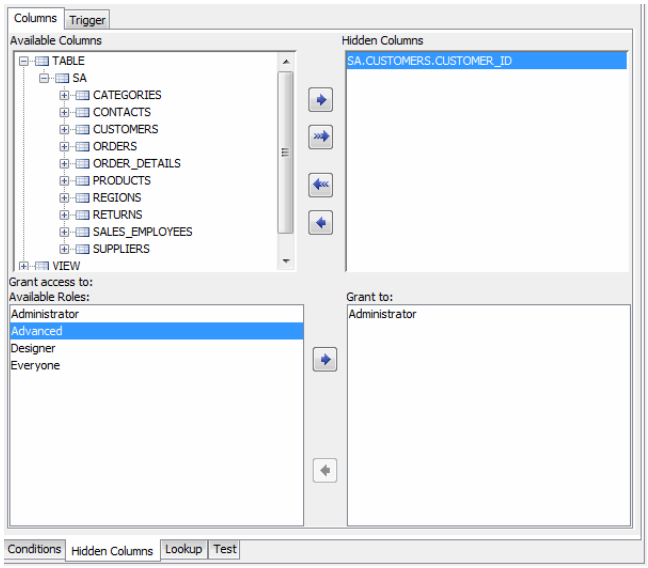
2. Expand the data source that contains the VPM you want to edit. Double-click the VPM to open it for editing.

3. In the VPM edit view, click the **Hidden Columns** tab at the bottom of the window.

4. Drag the desired columns from the ‘Available Columns’ pane to the ‘Hidden Columns’ pane.

   The columns listed in the ‘Hidden Columns’ pane are automatically hidden from all users, except for those who are associated with at least one of the roles listed in the ‘Grant to’ pane.

5. Add the desired roles to the ‘Grant to’ pane by clicking the arrow button.



6. Click 'Save' in the Style Studio toolbar to save the changes to the VPM.

Once the hidden columns are specified, the tables containing those columns are automatically listed in the **Lookup** tab. This indicates that the VPM security will be applied to any query that contains those tables.

A column that is derived from one or more hidden columns by an expression will be accessible only to users who have permissions for all of the columns used in the expression. However, for unparsable queries, derived columns should be explicitly tagged; see *VPM Filtering of Unparsable Queries*.

3.6.4 Trigger Scripts

The restrictions specified in a VPM are global by default and apply to all users. *Triggers* allow you to incorporate business logic into this cell-level security model. There are three types of triggers: *Lookup Trigger*, *Column Trigger*, and *Condition Trigger*. Trigger scripts use standard JavaScript syntax, and the following parameters are available to all three triggers:

PARAMETER	VALUE
user	The user id
roles	Array of roles that this user is associated with
groups	Array of groups that this user is associated with
parameter	Report parameters
tables	List of tables in the query
columns	Array of columns included in the query

Lookup Trigger

All tables for which the VPM specifies conditions or hidden columns listed under the **Lookup** tab. When a report or dashboard runs a query, the VPM will be applied to the query if the query makes referenced to any one of these “lookup” tables.

To apply the VPM to a query based on information other than its constituent tables, you can specify this logic in the lookup **Trigger** tab script. The lookup trigger script is executed only if the VPM would *not* have been applied otherwise. The VPM is applied only if the lookup trigger returns `true`.

Column Trigger

The hidden column **Trigger** allows you to modify the list of hidden columns for individual users, or for a specific role or a group of users. The following additional parameters are available for the column trigger:

PARAMETER	VALUE
hiddenColumns	Array of hidden columns defined in the VPM

The hidden columns trigger should return a modified array of hidden columns in the last statement of the script. For example:

```
hiddenColumns[hiddenColumns.length] = 'SA.CUSTOMERS.CITY';
hiddenColumns;
```

Condition Trigger

The condition **Trigger** tab enables you to modify the set of conditions specified in the VPM. The following additional parameters are available for the condition trigger:

PARAMETER	VALUE
condition	Condition string of the conditions defined in the VPM

The condition trigger should return the modified list of conditions in the last statement of the script. For example:

```
if(condition.length > 0) {
  condition += " and ";
}

condition += "SA.CUSTOMERS.STATE = 'CA'";
condition;
```

3.6.5 VPM Example

Walkthrough

In the following example, you will create a VPM that implements the following restrictions:

- The ‘CUSTOMER_ID’ column in the ‘CUSTOMERS’ table will be hidden from all users except those with the ‘Administrator’ role.
- Information about California customers will be hidden from all users except those with the ‘CASalesDept’ role.

A security provider is required in order to use VPM security. See *Specifying a Security Provider* in *Advanced Server Administration* for more information.

To begin, create a new user and role in your security model. (If you are using default security provider, do this from within Enterprise Manager. See *Defining Users, Roles, and Groups* for instructions.)

1. Create a new role called “CASalesDept”. Users who are assigned this role will have access to information about all of the customers in the state of California.

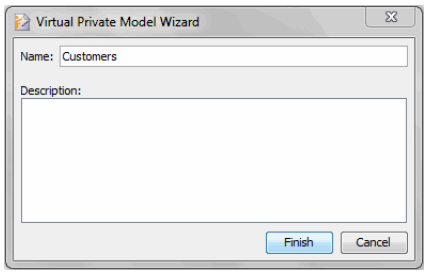
2. Create a new user “Eric,” and assign him the role of “CASalesDept”.

You will now define a VPM that operates on the ‘CUSTOMERS’ table. It will hide the ‘CUSTOMER_ID’ field from all users, and hide California customer information from unauthorized users.

3. Open the **Asset** panel, and expand the ‘Data Source’ node.

4. Right-click the ‘Orders’ node, and select ‘Add Virtual Private Model’ from the context menu. This opens the ‘Virtual Private Model Wizard’.

5. Name the new VPM “Customers”. Notice that a short description for the VPM can also be added in this wizard.

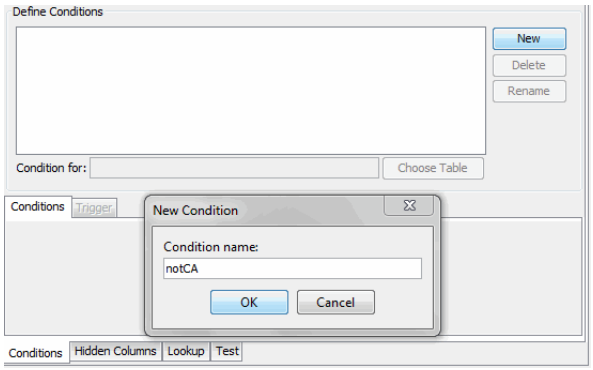


6. Click on the ‘Finish’ button. This opens the new VPM for editing.

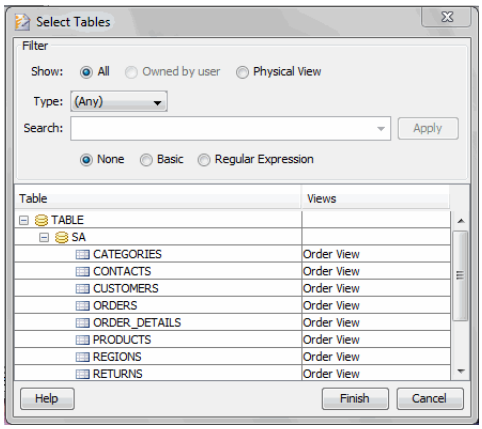
7. Click the **Conditions** tab at the bottom of the VPM editing panel.

8. Click on the ‘New’ button. This opens the ‘New Condition’ dialog opens.

9. Name the new condition “NotCA” and click on the ‘OK’ button.



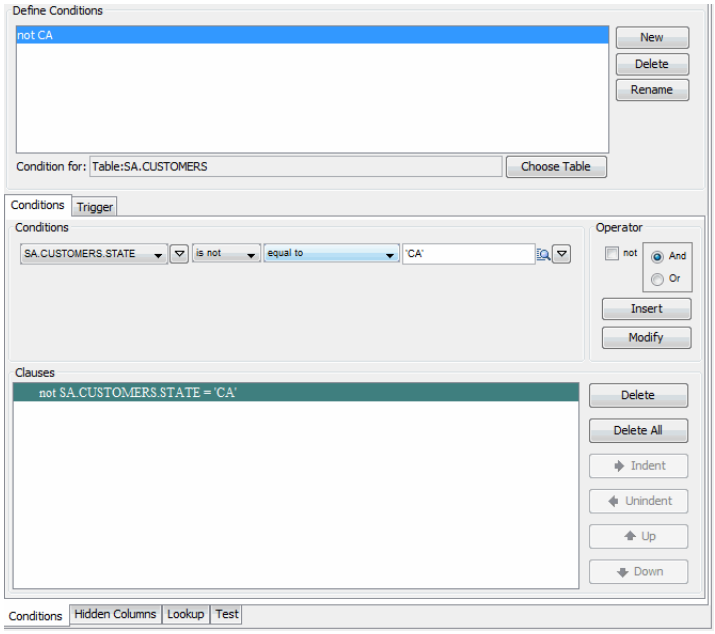
10. Click the ‘Choose Table’ button. This opens the ‘Select Table’ dialog box.



11. Select the ‘SA.CUSTOMERS’ table and click on the ‘Finish’ button.

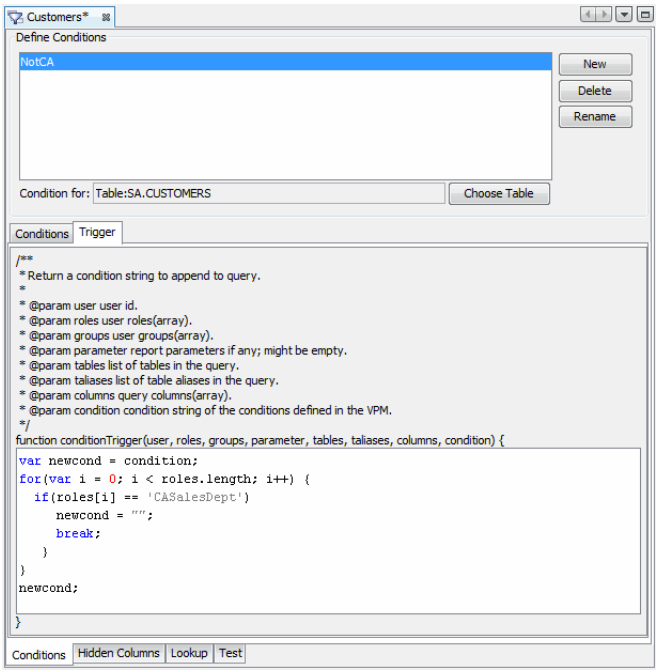
12. Specify the following condition and click the ‘Append’ button:

```
[SA.CUSTOMERS.STATE] [is not] [equal to] ['CA']
```



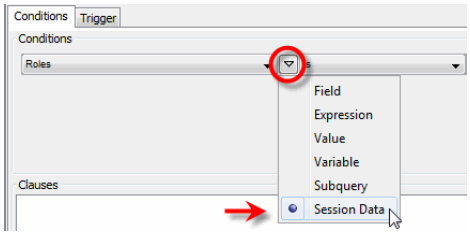
13. Click the **Trigger** tab and add the following script:

```
var newcond = condition;
for(var i = 0; i < roles.length; i++) {
  if(roles[i] == 'CASalesDept') {
    newcond = "";
    break;
  }
}
newcond;
```



This script tests all of the roles assigned to the current user. If one of these roles is 'CASalesDept', then the condition specified earlier (and provided to the script as variable `condition`) is replaced with an empty string, indicating a null condition. The effect is that no state filtering occurs for a user with role 'CASalesDept'.

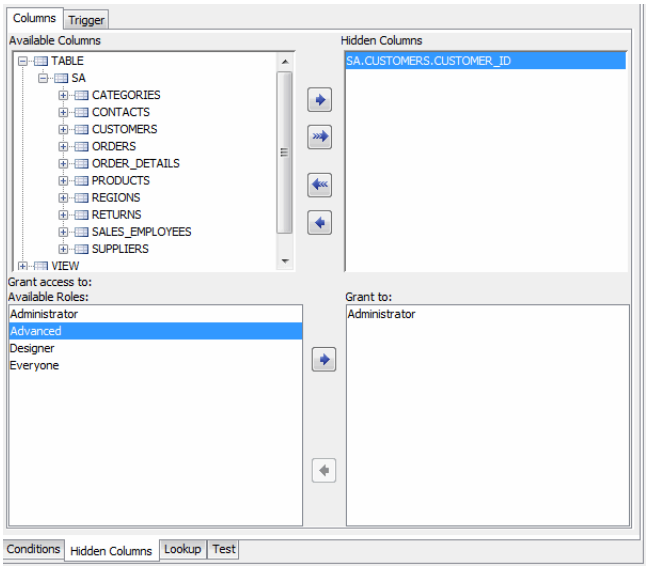
To achieve the same result *without* using a trigger script, join ("and") the following clause to the original condition: `[Roles] [is not] [equal to] ['CASalesDept']`. (To select the 'Roles' option in the left-menu, first choose the 'Session Data' option from the adjacent pop-up menu.)



14. Select the **Hidden Columns** tab.

15. Select 'SA.CUSTOMERS.CUSTOMER_ID' field from the list of 'Available Columns' and click on the right-arrow button to add the field to the list of 'Hidden Columns'.

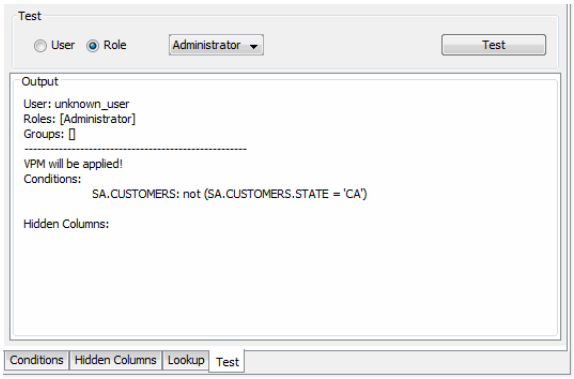
16. Select 'Administrator' from the list of 'Available Roles' under 'Grant access to'. Click the right-arrow button to grant permission to the 'SA.CUSTOMERS.CUSTOMER_ID' field to all users having role 'Administrator'.



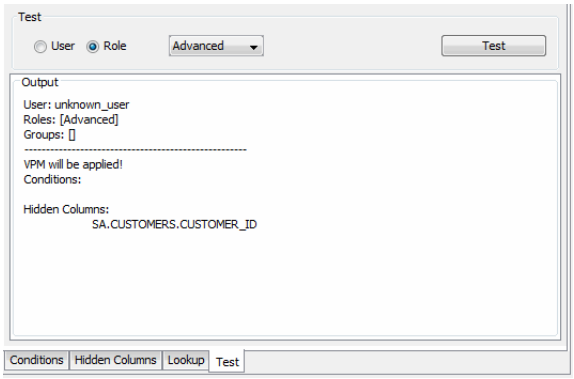
17. Click the **Lookup** tab (at bottom) and notice that the 'SA.CUSTOMERS' table is included in the list of lookup tables. All tables having conditions or hidden columns will appear in this list.

18. Click the **Test** tab (at bottom), and select 'Role'.

19. Select 'Administrator' from the list of roles and click on the 'Test' button. Users with 'Administrator' role do not see California data because of the 'CUSTOMERS.STATE' condition. However, because no hidden columns are listed here, they are able to see the 'CUSTOMERS.CUSTOMER_ID' field.

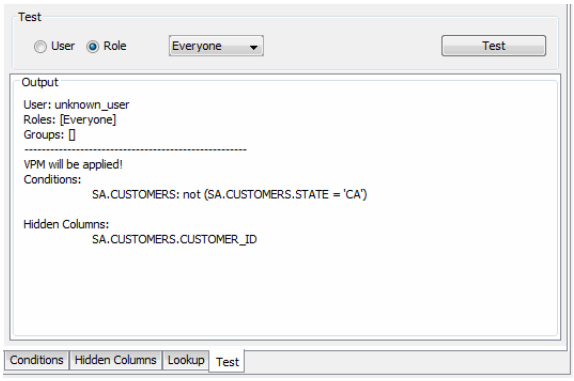


20. Now select 'CASalesDept' from the list of roles and click on the 'Test' button. Users belonging to this role do not have the 'CUSTOMERS.STATE' condition applied to their query. This is because of the condition trigger script. However, the 'CUSTOMERS.CUSTOMER_ID' column is hidden from all of these users.



21. Select the 'User' button, and choose 'Eric' from the user list. Click 'Test' and confirm that Eric's permissions are the same as those for the role of 'CASalesDept' because Eric has been assigned this role.

22. Select the ‘Role’ button again. Choose ‘Everyone’ from the role list and click on the ‘Test’ button. Notice that the ‘CUSTOMERS.STATE’ condition as well as the ‘CUSTOMERS.CUSTOMER_ID’ hidden column are applied to queries from users having this role.





23. Click on the ‘Save’, button on the Style Studio toolbar to save the changes made to this VPM.

3.6.6 VPM Filtering of Unparsable Queries

In some cases, Style Studio may not be able to parse highly complex SQL queries, or those that contain custom database syntax. If a query is unparsable, Style Studio cannot automatically identify the columns being returned by the query, and therefore cannot automatically apply the VPM security model.

To enable VPM filtering for an unparsable query, you must embed special tags into the query’s SQL string. These tags supply information to Style Studio about hidden columns and filtering conditions. The special tags are listed below.

TAG	REFERENCE
`--vpm.tables`	A list of the tables contained in the query (to be used for VPM filtering).
`--vpm.columns`	A list of the hidden columns used in the query.
`--vpm.aliases`	A list of the table aliases used in the query.
`/*<idx>*/`	The integer index of the hidden column in the ‘vpm.columns’ list from which the tagged query column should inherit security. (Numbering begins with 1.)
`/*<where>*/`	Denotes the “where” clause of the query for which filtering conditions are to be added by the VPM

The ‘`vpm.tables`’ and ‘`vpm.columns`’ tags should be added at the beginning of the unparsable query. The ‘`vpm.tables`’ tag is followed by a list of tables used in the query. The ‘`vpm.columns`’ tag is followed by a list of table

columns from which the query columns will inherit VPM settings. Every item on this list must exactly match one of the columns specified in the 'Hidden Columns' panel of the **Hidden Columns** tab. (See Step 14 of the *VPM Example* section.)

To specify the security for a particular returned query column, you need to associate the given query column with one of the hidden columns in the '`vpm.columns`' list. To do this, wrap the query column inside a pair of '`/*<idx>*/`' tags, and set '`idx`' to the index of the listed column from which the query column should inherit security.

If the VPM defines filtering conditions for any one of the tables in the '`vpm.tables`' list, those conditions will be applied within the '`/*<where>*/`' tags.

Example: Filtering Unparsable Queries

Consider the following query.

```
select
SA.CUSTOMERS.COMPANY_NAME,SA.ORDER_DETAILS.QUANTITY+10,
SA.ORDERS.DISCOUNT,SA.PRODUCTS.PRODUCT_NAME,SA.PRODUCTS.PRI
CE,SA.PRODUCTS.DESCRIPTION
from SA.CUSTOMERS, SA.ORDER_DETAILS, SA.ORDERS, SA.PRODUCTS
where
SA.ORDER_DETAILS.PRODUCT_ID = SA.PRODUCTS.PRODUCT_ID and
SA.ORDERS.ORDER_ID = SA.ORDER_DETAILS.ORDER_ID and
SA.ORDERS.CUSTOMER_ID = SA.CUSTOMERS.CUSTOMER_ID
```

Assume that the query is unparsable, and that you want to apply security to the '`CUSTOMERS.COMPANY_NAME`' and '`ORDER_DETAILS.QUANTITY+10`' columns returned by the query, and that you also want to apply a VPM filter to the '`where`' clause.

To configure the above security settings, follow these steps:

1. In the **Hidden Columns** tab, add the '`SA.CUSTOMERS.COMPANY_NAME`' and '`SA.ORDER_DETAILS.QUANTITY`' columns to the 'Hidden Columns' list.

2. Specify the desired VPM filtering conditions in the **Conditions** tab.

3. Open the query in Style Studio, and click the **SQL String** tab to edit the query SQL as follows.

4. Place the '`/*<where>*/`' tags around the "`where`" clause of the query:

```
/*<where>*/SA.ORDER_DETAILS.PRODUCT_ID =
SA.PRODUCTS.PRODUCT_ID and SA.ORDERS.ORDER_ID =
SA.ORDER_DETAILS.ORDER_ID and SA.ORDERS.CUSTOMER_ID =
SA.CUSTOMERS.CUSTOMER_ID/*</where>*/
```

5. Add the following tags at the beginning of the query:

```
--vpm.tables:SA.CUSTOMERS,SA.ORDER_DETAILS,SA.PRODUCTS
--
vpm.columns:SA.ORDER_DETAILS.QUANTITY,SA.CUSTOMERS.COMPANY_
NAME
```

   Filtering conditions defined by the VPM on the specified tables (`vpm.tables`) will be applied within the '`/*<where>*/`' tags. The specified columns (`vpm.columns`) will be referenced by the '`/*<idx>*/`' tags in the queries, as follows.

Note: The SQL query *must remain valid* when the material enclosed by tags is removed. Commas should therefore be placed *inside* the tags. Line breaks may not occur between two special tags.

6. Add '`/*<idx>*/`' tags around the query columns that will inherit security:

```
select /*<2>*/SA.CUSTOMERS.COMPANY_NAME,/*</2>*/
/*<1>*/SA.ORDER_DETAILS.QUANTITY+10,/*</1>*/ ...
```

   Note that the index 'idx' refers to the location of the corresponding hidden column in the 'vpm.columns' list. Therefore, the tag '`/*<1>*/`' refers to the first item in the 'vpm.columns' list, which is '`ORDER_DETAILS.QUANTITY`'. This means that the returned column '`SA.ORDER_DETAILS.QUANTITY+10`' inherits security from the hidden column '`ORDER_DETAILS.QUANTITY`'.

The complete tagged query is shown below:

```
--vpm.tables:SA.CUSTOMERS,SA.ORDER_DETAILS,SA.PRODUCTS
--
vpm.columns:SA.ORDER_DETAILS.QUANTITY,SA.CUSTOMERS.COMPANY_
NAME
select /*<2>*/SA.CUSTOMERS.COMPANY_NAME,/*</2>*/ /*<1>*/
SA.ORDER_DETAILS.QUANTITY+10,/*</1>*/
SA.ORDERS.DISCOUNT,SA.PRODUCTS.PRODUCT_NAME,SA.PRODUCTS.PRI
CE,SA.PRODUCTS.DESCRIPTION
from SA.CUSTOMERS, SA.ORDER_DETAILS, SA.ORDERS, SA.PRODUCTS
where /*<where>*/SA.ORDER_DETAILS.PRODUCT_ID =
SA.PRODUCTS.PRODUCT_ID and SA.ORDERS.ORDER_ID =
SA.ORDER_DETAILS.ORDER_ID and SA.ORDERS.CUSTOMER_ID =
SA.CUSTOMERS.CUSTOMER_ID/*</where>*/
```

¢

3.7 OLAP Overlay – Multidimensional Analysis

`Note: OLAP features are available only in Style Intelligence.`

Style Studio provides a lightweight, logical mapping tool called OLAP overlay to allow direct "slice-and-dice" on ER schema data. The OLAP overlay is an optional component of a data model that provides flexible ways to dynamically group, aggregate, and display summary information.

An OLAP overlay defines attributes as measures and dimensions. *Measures* are numeric values that are additive in nature. For example,

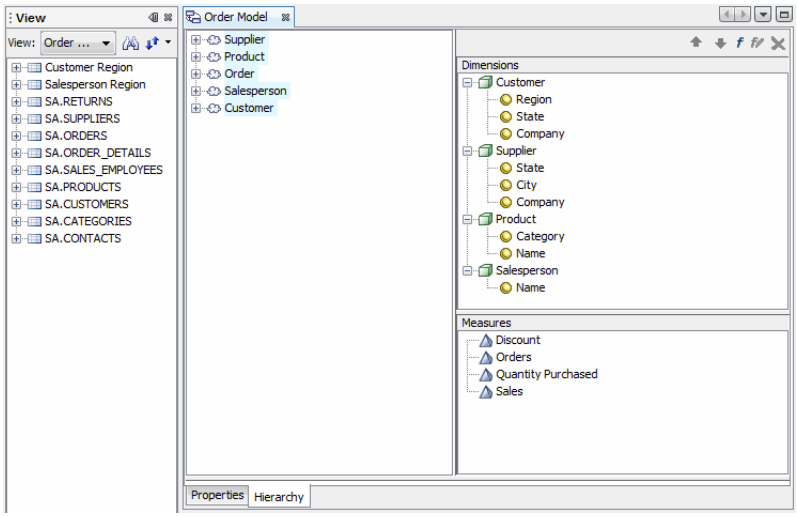
‘order sale amount’ fits this definition because adding all ‘order sale amounts’ will give the ‘sales total’. Bank account ‘daily balance’, on the other hand, is not additive because adding two days of a balance does not provide meaningful information. Therefore, account balance is considered semi-additive, because it can still be averaged for useful purposes.

Dimensions are the descriptive parameters for measures that can be used to filter and group. For example, ‘purchase date’, ‘ship date’, ‘product’ and ‘sale location’ are all possible dimensions for orders. Dimensions also often form natural hierarchies, i.e., a product belongs to a category. This allows business users to easily use different combinations of dimensions and measures to analyze performance.

The total collection of dimensions and measures represents a multidimensional data structure known as a *data cube*. Because of its organization into dimensions and measures, conceptually, the OLAP overlay can also be considered a *star schema*.

3.7.1 Creating a Hierarchical OLAP Overlay

The **Hierarchy** tab of the logical model provides an interface to create the OLAP overlay. Entities and attributes are listed in the center pane in a tree structure.



To add *dimensions* to the overlay, do one of the following:

The order of dimensions within each dimensional hierarchy is significant, as this is the order of the drilling hierarchy.

- Drag an individual attribute from the logical model (center panel) to the ‘Dimensions’ panel. This adds the attribute as a dimension.
- Drag an entire entity from the logical model (center panel) to the ‘Dimensions’ panel. This adds all attributes of the entity as a dimensional hierarchy.

To add a *measure* to the overlay, drag an attribute from the logical model (center panel) into the ‘Measures’ area.

When you create dimensions, give special attention to Date fields. Breaking a single date field into individual components can drastically increase usability for the end user. Additionally, mapping date components into business attributes can be a powerful usability enhancement, for example, mapping the month of a date field so that June becomes “calendar month 1,” and so on.

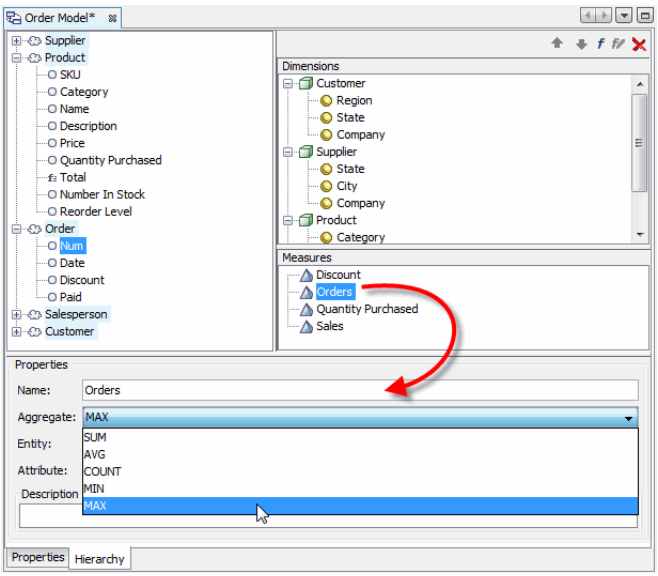
To create new date components of this kind, you can define expression attributes in the **Model** tab. These expression attributes will be available for use in the hierarchical overlay. For example, to create a new ‘Month’ attribute in which June is “calendar month 1,” you could define a new expression attribute with the formula

```
Month(DateAdd('m',-5,field['orders.order_date']))
```

where ‘`orders.order_date`’ is the existing date column. The particular formula would need to use SQL syntax and functions that are appropriate to the underlying database. See *Defining Expression Attributes* for more information.

3.7.2 Editing Dimension and Measure Properties

You can change the names of dimensions and measures from the ‘Properties’ panel of the **Hierarchy** tab. Additionally, the aggregation method for each measure can be specified using the ‘Aggregate’ menu.



The aggregation method determines how a particular measure is summarized at the different levels of dimensional granularity. For example,

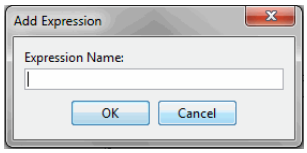
consider the case where you have the hierarchically-related dimensions ‘Month’ and ‘Day’, and a measure ‘Cost’. If you aggregate the measure by ‘AVG’, then the value of the measure when viewed along the ‘Month’ dimension will be “average cost per month,” while the value of the measure when viewed along the ‘Day’ dimension will be “average cost per day.” Likewise, if you aggregate the measure by ‘MIN’, then the value of the measure when viewed along the ‘Month’ dimension will be “minimum cost per month,” while the value of the measure when viewed along the ‘Day’ dimension will be “minimum cost per day.”

3.7.3 Creating Derived Measures

In addition to selecting existing model attributes as measures for the hierarchical overlay, you can create new “derived measures,” which are measures computed from existing attributes.

To create a derived measure:

1. Click the ‘Create derived measure’ button above the ‘Dimensions’ tree in the **Hierarchy** tab. This opens the ‘Add Expression’ dialog box.



2. Enter a name for the new measure and click ‘OK’. This opens the Script Editor.

3. In the Script Editor, enter an expression to define the new measure. The expression should use the appropriate SQL syntax for the underlying database. (There are no aggregation functions available for measure definition because aggregation is specified in the ‘Properties’ panel of the **Hierarchy** tab.)

4. Click the ‘Save and Close’ button. The new derived measure will appear alongside the existing measures.

3.7.4 Example: OLAP Overlay

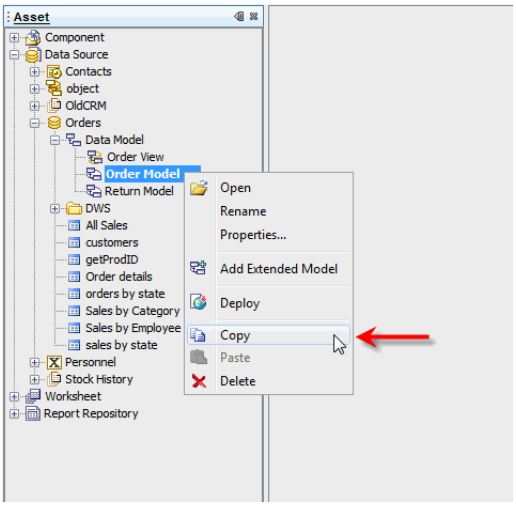
Walkthrough

In this example, you will create a data cube (OLAP overlay) with two pairs of nested dimensions and two measures. This overlay will be based on the existing Order Model, so first make a working copy of that logical model:

1. Open the **Asset** panel, and expand the ‘Data Source’ node.

2. Expand the ‘Orders’ node, and expand the ‘Data Model’ node.

3. Right-click the ‘Order Model’ node, and select ‘Copy’ from the menu.



4. Right-click on the ‘Data Model’ node, and select ‘Paste’ from the context menu. This creates a new data model called ‘Copy of Order Model’.

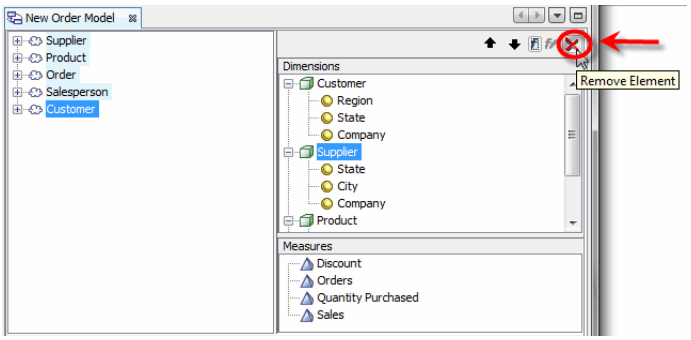
5. Right-click this copy and select ‘Rename’. Rename the copy ‘New Order Model’ and click ‘OK’.

6. Double-click the ‘New Order Model’ node to open the copied model.

7. Click the **Hierarchy** tab at the bottom of the edit panel.

8. Remove the existing hierarchical overlay, so that you can start fresh. To do this, follow the steps below:

    a. Select a dimension (cube icon) in the ‘Dimensions’ panel. Click the ‘X’ button to delete the entity from the overlay.



b. Repeat the above step to delete all additional dimensions in the ‘Dimensions’ panel.

c. Repeat the above steps to delete all measures in the ‘Measures’ panel.

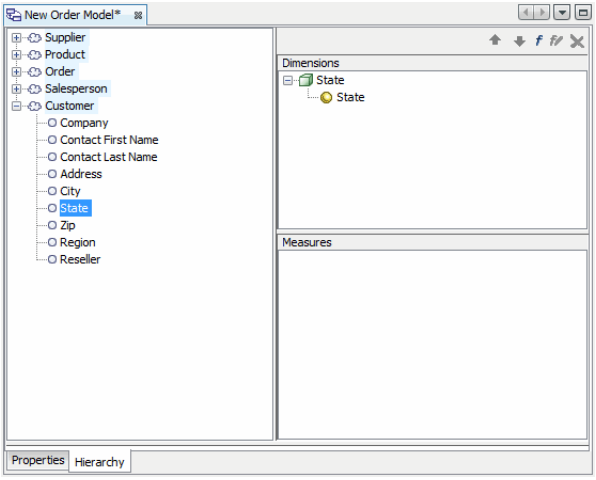
You will now specify dimensions and measures for the new hierarchical overlay. The four dimensions are as follows:

- ‘Customer State’
- ‘Customer City’
- ‘Product Category’
- ‘Product SKU’

The first pair forms a hierarchical group, i.e., ‘City’ represent a finer granularity than ‘State’. The second pair also forms a hierarchical group, i.e., ‘SKU’ represents a finer granularity than ‘Category’.

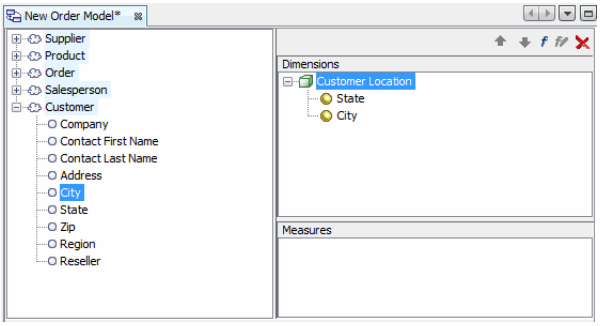
First you will create the dimensional hierarchy representing customer location. This hierarchy will be composed of ‘Customer State’ and ‘Customer City’:

9. Click and drag the ‘Customer.State’ attribute from the model panel to the ‘Dimensions’ panel. This adds a new dimension called ‘State’ to the ‘Dimensions’ panel, automatically placed under a hierarchy labeled ‘State’.



10. Click on the ‘State’ hierarchy node (cube icon) to select it. In the ‘Name’ field of the ‘Properties’ panel, enter the name “Customer Location.” (You can also enter a description of this hierarchy, if desired.)

11. Click and drag the ‘Customer.City’ icon from the model panel to the ‘Dimensions’ panel. Drop it on top of the ‘Customer Location’ node.



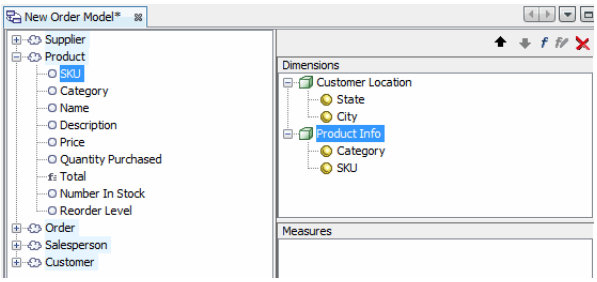
The ‘City’ attribute should now appear as a dimension in the ‘Customer Location’ hierarchy. Make sure the dimension order is correct: Finer-granularity dimensions should appear below the corresponding courser-granularity dimensions; thus, ‘City’ should appear below ‘State’. If the ordering is wrong, move the dimension by using the arrow buttons above the panel.



12. To add the next hierarchy, drag the ‘Product.Category’ attribute from the model panel to an empty location in the ‘Dimensions’ panel. This adds a new dimension called ‘Category’ to the ‘Dimensions’ panel, automatically placed under a hierarchy labeled ‘Category’.

13. Click on the ‘Category’ hierarchy root node (cube icon) to select it. In the ‘Name’ field of the ‘Properties’ panel, enter the name “Product Info.” (You can also enter a description of this hierarchy.)

14. Click and drag the ‘Product.SKU’ icon from the model panel to the ‘Dimensions’ panel. Drop it on top of the ‘Product Info’ node.



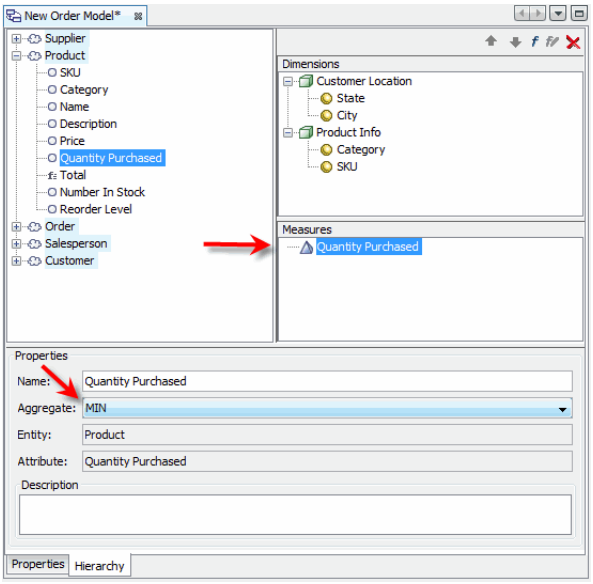
This adds the ‘SKU’ attribute as a dimension in the ‘Product Info’ hierarchy. Again, make sure the dimension ordering is correct, ‘SKU’ below ‘Category’.

Now you will add two measures. The first measure will be the quantity of product purchased, and the second measure will be the dollar amount of discounts applied. Because this second measure is not available as an existing attribute, you will derive it from other attributes by using an expression.

15. Click and drag the ‘Product.Quantity Purchased’ attribute from the model panel to the ‘Measures’ panel. This adds a new measure called ‘Quantity Purchased’ to the ‘Measures’ panel.

16. Click to select the ‘Quantity Purchased’ measure in the to the ‘Measures’ panel. This opens the ‘Properties’ panel at bottom.

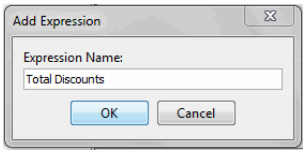
17. From the ‘Aggregate’ menu, select the ‘MIN’ option. This will compute the minimum quantity purchased for each level of the dimension.





18. To define the derived measure, the dollar amount of discounts, click the ‘Create Derived Measure’ button above the ‘Dimensions’ tree. This opens the ‘Add Expression’ dialog box.

19. Enter “Total Discounts” in the ‘Expression Name’ field. This will be the name of the new measure.



20. Click the ‘OK’ button in the ‘Add Expression’ dialog box to open the Script Editor.

You can double-click an attribute names in the ‘Fields’ tree to add it to the expression.

21. In the Script Editor, enter the following expression to define the new measure representing the total dollar amount of discounts:

```
field['Product.Total'] * field['Order.Discount']
```



22. Click the ‘Save and Close’ button. This adds the new derived measure ‘Total Discounts’ alongside the other measure.

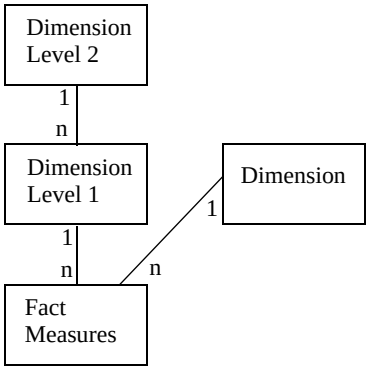
23. From the ‘Aggregate’ menu in the ‘Properties’ panel, select the ‘SUM’ option. This will compute the summed discount in dollars for each level of dimension granularity.

The hierarchical overlay is now fully specified, and you can use this OLAP model when you design Viewsheets.

See Also
Creating a Crosstab, in *Advanced Dashboard Design*.

3.7.5 Design Considerations

You should give careful consideration to designing a properly formed OLAP overlay. In particular, the relationships between dimensions and measures, and between attributes within a dimension, should have the cardinality pattern shown below:



This cardinality pattern will avoid “traps” and ensure that aggregation performed by end users gives correct results.

In the event that you need to implement your OLAP design in a physical database (rather than using the hierarchical overlay), the following books will provide helpful design ideas:

- **The Data Warehouse Toolkit**, by Ralph Kimball
- **Building the Data Warehouse**, by William Inmon

4 Non-Relational Data Sources

The following sections explain how to configure various non-relational data sources.

4.1 Generic Multidimensional Data Source

Multidimensional databases organize data into cubes containing measures and dimensions. This technology is often referred as MOLAP. The primary benefit of using MOLAP is the enhanced performance resulting from pre-aggregation.

Style Intelligence uses the XMLA standard to query a Cube engine with MDX queries, and automatically creates models for these multidimensional databases.

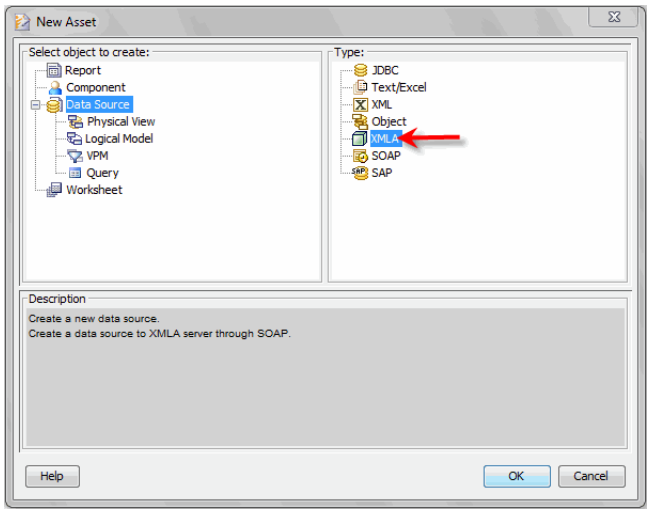
```
Recommendation: Within a given level of a dimension, do not create
multiple entries with the same name. This creates an ambiguity for
Viewsheet selections.
```

To configure a MOLAP data source, follow the steps below:

Cubes can be visualized only within Viewsheets. All available cubes are automatically available in the **Component** panel.

1. Press the ‘New’ button in the Style Studio toolbar. This opens the ‘New Asset’ dialog box.

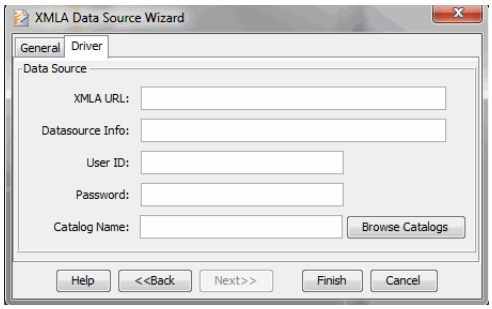
2. In the left panel, select the ‘Data Source’ node. In the right ‘Type’ panel, select ‘XMLA’ and press ‘OK’.



   This opens the ‘XMLA Data Source Wizard’ dialog box.

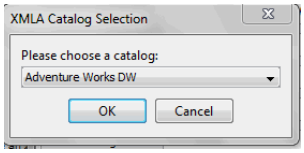
3. In the ‘XMLA Data Source Wizard’ dialog box, select the **General** tab and enter the data source ‘Name’ and ‘Description’.

4. Select the **Driver** tab and enter the 'XMLA URL'.

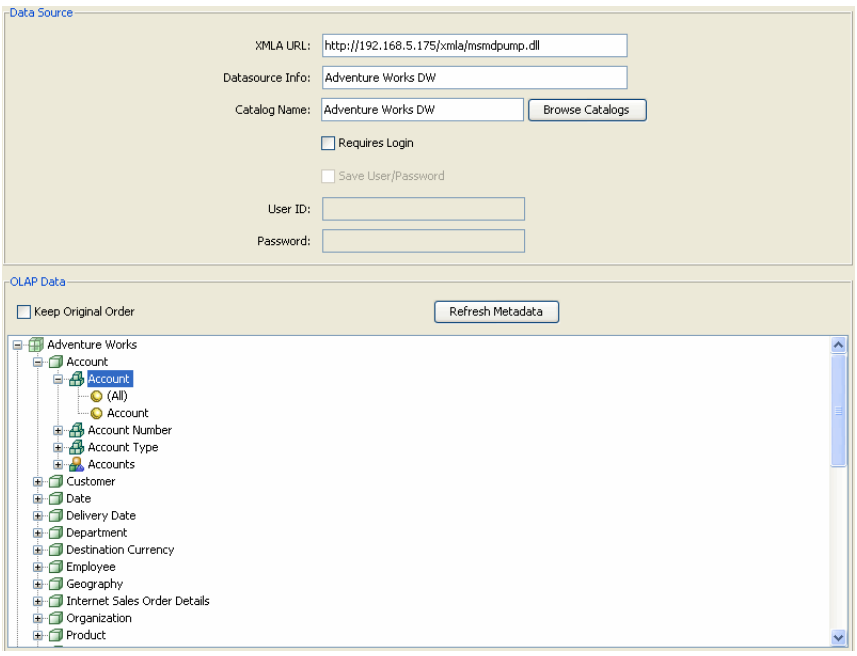


XMLA is used to communicate with the OLAP server. The URL is the path to the XMLA service; for example, the URL for SQL Server is the file *msxisapi.dll*, which is in the *Isapi* directory of the XMLA installation.

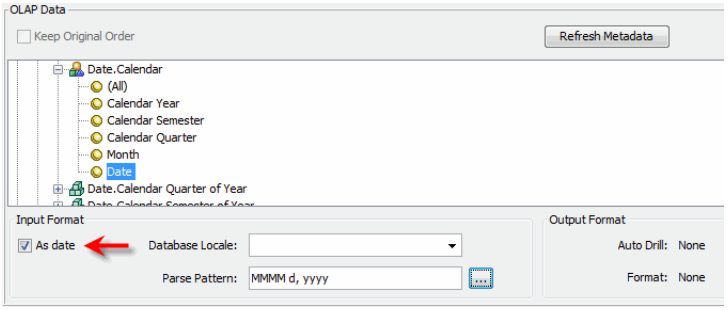
5. Press the 'Browse Catalogs' button and select the database catalog name from the menu. Press 'OK'.



6. Press 'Retrieve Metadata' to load the schema. This button can also be used to update the schema if it is changed at a later time.



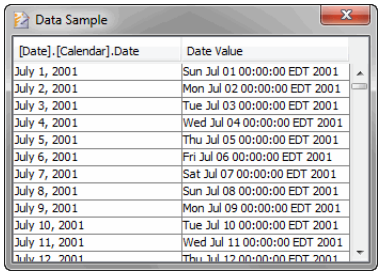
7. (Optional) For a date dimension, select a desired level of the date dimension in the tree, and enable the 'As Date' option. In the 'Parse Pattern' field, enter the date format corresponding to the date values stored in the data source.



The 'As Date' option allows the date field to be displayed in Charts, Crosstabs, and Selection Lists with normal date-sorting features, and allows the date field to be filtered by a Range Slider component (but not by a Calendar component).

8. Select 'Keep Original Order' to display dimension values within a chart or a crosstab in the same order as retrieved from the data source.

9. Press the 'View Sample Data' button to verify that the dates are being parsed correctly.





10. Press 'Save' on the toolbar to save the data source.

See Also

Appendix B: *OLAP Server Setup*, for configuration details about specific databases.

4.2 SQL Server Analysis Services

To create a datasource for SQL Server Analysis Services (SSAS) cubes, follow the procedure described in *Generic Multidimensional Data Source*.

Microsoft provides an ISAPI called *msmdpump.dll* that can be leveraged via IIS to query the Analysis Server as a web service. You can find this file

in your Analysis Services installation: *Microsoft SQL Server\<instance>\OLAP\bin\isapi*. For detailed instructions on how to set up IIS, see the Microsoft Developer Network.

The IIS server can run in any of the following environments:

- The physical host running analysis services (quickest and simplest set-up). Note: If IIS is *not* running on the same machine as the Analysis Server, you will also need to install the OLE DB for SQL Server on the IIS host machine (SQL Server 2008 or SQL Server 2012).
- The physical host running Style Intelligence
- A completely separate machine

Security for the IIS application (virtual directory) should be set to Anonymous or Basic.

4.3 XML Data Source

Style Intelligence uses XQuery for querying XML documents and other forms of hierarchical data. XQuery transforms the hierarchy into a flat tabular format to facilitate reporting and analysis. The conditional expression syntax is based on SQL. See Appendix A: *Hierarchical Query Condition Expression Syntax* for more information.

See Also
Querying XML or Web Service Data Source, for information on creating a query.

4.3.1 Defining an XML Data Source

The following example illustrates how to configure an XML data source.

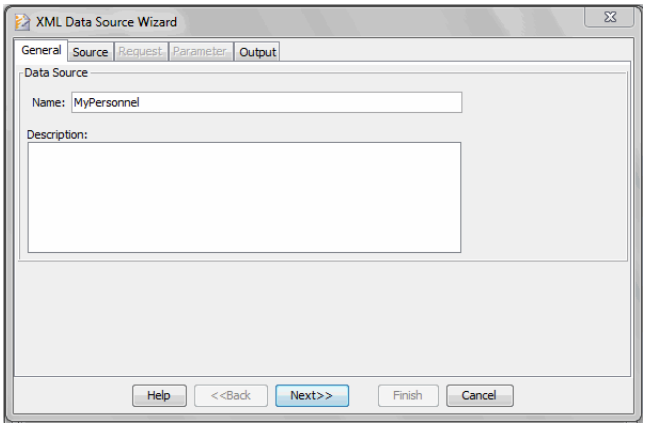
Walkthrough

To create an XML data source named “MyPersonnel,” follow the steps below:

1. Press the ‘New’ button in the Style Studio toolbar. This opens the ‘New Asset’ dialog box.

2. In the left panel, select the ‘Data Source’ node, and in the right ‘Types’ panel select ‘XML’. Press ‘OK’. This opens the ‘XML Data Source Wizard’.

3. Select the **General** tab. Enter “MyPersonnel” for the data source ‘Name’.

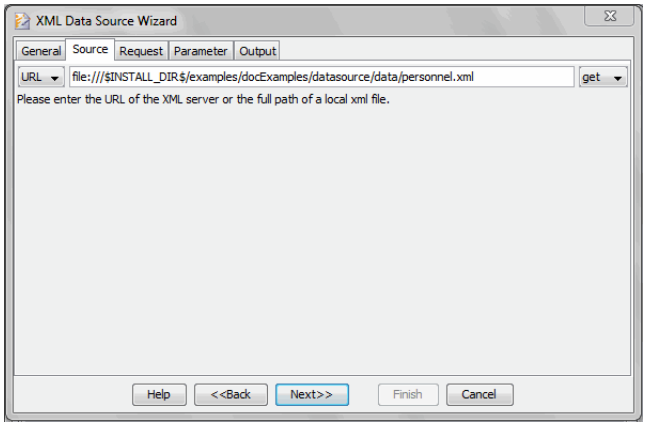


4. Select the **Source** tab. Enter a data source URL or file path. (For this example enter "`{Installation}/examples/docExamples/datasource/data/personnel.xml`", where `{Installation}` is the Style Intelligence installation directory, for example "`C:\StyleIntelligence`".)

   An XML data source can get data from any URL or file. In particular, it can communicate with an HTTP server using the POST or GET methods to pass parameters to the server side program. A DTD or XML schema must be specified for the XML data stream.

   You can use a variable in the URL field, e.g., '`C:/datafiles/$(parameter_name)`' to make the data source URL dynamic in part or whole. Parameters can be specified as system variables, as report parameters within the *sree.properties* file, or as URL parameters. The report engine will look for the parameters in the above sequence, and if not found, will default to prompting the user for the parameter.

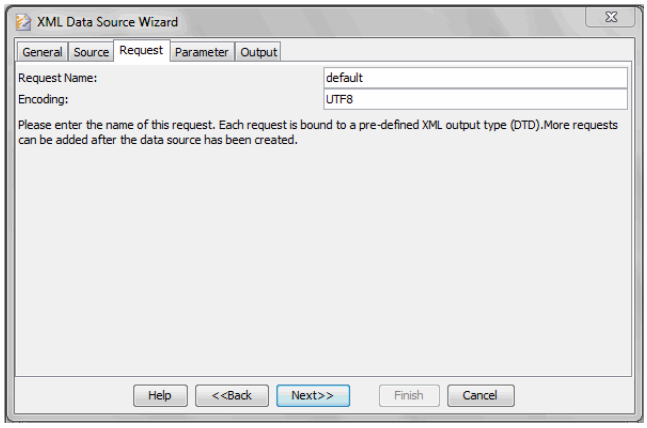
5. Select the HTTP method. If the URL does not use HTTP, select the GET method.



6. Select the **Request** tab.

7. Enter the name and the encoding of the request, "`default`" and "`UTF8`".

   Each request is bound to a pre-defined XML output type (DTD). More requests can be added after the data source has been created. A request corresponds to a particular set of parameters in the data source server. For example, a servlet may expect a 'file' parameter which contains the file name of the XML file on the server.

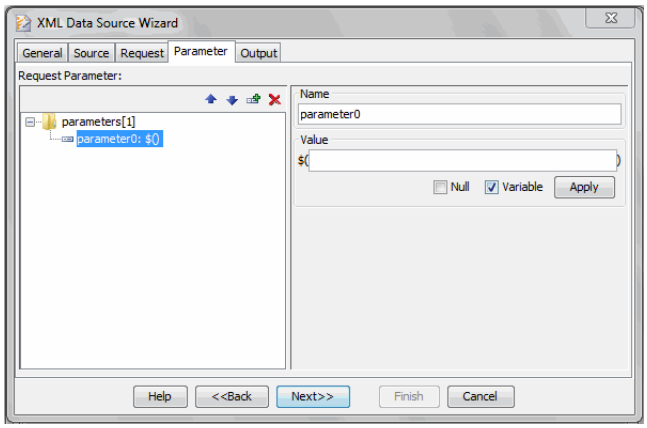


8. Select the **Parameter** tab.

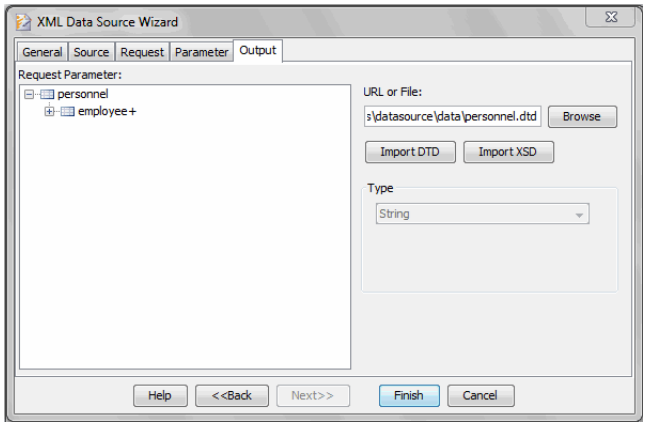
   The **Parameter** tab shows the current parameters for the request. For HTTP requests, the parameters are ordered in a parameter tree. The root of the tree is the 'Parameters' node. The child of the root is a sequence node, with one or more pairs of parameters. Each pair consists of a name and a value node. The name node contains the name of the HTTP request parameter, and the value node contains the value of the parameter.

Check the ‘Variable’ box if you want the parameter value to be a variable.

9. To add a new parameter, select the ‘`parameters[0]`’ node and click on the ‘Add parameter’ button. Enter the name of the parameter and its value. Repeat to add more parameters.



10. Select the **Output** tab. Press ‘Browse’ and choose the DTD or XSD file that corresponds to the XML file (In the example, *personnel.dtd*). Press ‘Import DTD’.



11. Press ‘Finish’. This closes the Wizard and creates the XML data source.

12. Select the **Output** tab. The output tree shows the XML schema defined in the DTD file. Because the DTD format does not specify the real type of the data in the XML elements, all elements are treated as strings by default.

    You do not ordinarily need to specify XML element types. This is only necessary if the element is used in the query selection conditions as a non-string type. For example, if the birthday is compared in a conditional expression with another date value, the

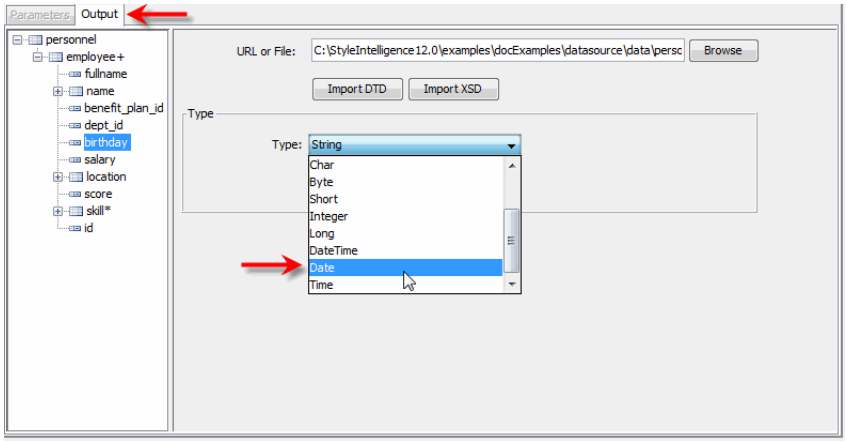
See *XML Element Types* for more information about the types.

type must be specified for the XML element, otherwise it will be treated as a string and the comparison will be meaningless.

Since only one schema is defined for each request, a request can only generate XML data in the specified schema. A separate request must be defined in the data source for each different XML output type.

13. (Optional) To manually override the element type for an element, follow the steps below:

    a. Expand the 'employee' node, select the 'birthday' node, and change its type to 'Date'.



    b. Select the 'salary' node on the tree, and change its type to 'Double'.

    c. Set any desired format for the field (e.g., `MMM-dd-yy` for a date or `###.00` for a number). See *Adding a Format to a Data Field* for more information about format specifications.

14. Press the 'Save' button in the Style Studio toolbar to save the new data source.

Every XML data source must contain at least one request. For more complex XML data sources, it is often necessary to create multiple requests. Each request can have different parameters and a different XML output type. If the URL is not HTTP-based or does not require any parameters, simply create a request without specifying any parameters.

4.3.2 XML Element Types

The XML Element Types can be specified as any one of the following pre-defined types:

– String
– Boolean
– Float
– Double
– Char
– Byte
– Short
– Integer
– Long
– TimeInstant – Date and time
– Date – Date only
– Time – Time of day

For Date and Time data types, you can supply a date format to create a date parser. Construct the format string according to the `java.text.SimpleDateFormat` specification. (See *Adding a Format to a Data Field* for more information about format specifications.) For example, to parse the date '`01/30/2000 01:30:59`', use the format '`MM/dd/yyyy hh:mm:ss`'. The '`double`' element type requires that the decimal format be specified. Construct the format according to `java.text.DecimalFormat` specification. For example, to add commas to a number, use '`#, ###, ###. ##`'.

4.4 Web Service Data Source

The example web service data source must run in a J2EE application server.

Style Studio allows you to create a data source that retrieves data from a web service with any standard SOAP implementation. If a working client can be generated by `wsimport` with no arguments or custom binding, you only need to provide the URL of the WSDL in order to create the web services data source. If you do not create a client stub, one will be created automatically by Style Studio. To create a stub manually, see *Creating the Client Stub*.

See Also
Querying XML or Web Service Data Source, for information on creating a query.

4.4.1 Defining a Web Service Data Source

Before you create any web service data source, verify the following points:

- The web service is running on the host server.
- (Optional) You have created a Java client stub as described in *Creating the Client Stub*.

The example below uses a web service called 'PeopleService' which is deployed with the default installation. To follow the example, use the *peopleservice-client.jar* file in the *sree/WEB-INF/lib* folder as the client

stub. Make certain that you add it to the Style Studio classpath. (See *Configuring the Style Studio Classpath*.)

Walkthrough

This example illustrates how to create a SOAP data source that connects to the 'PeopleService' web service (included in the default installation).

Before you start, verify the following:

- Start the server and make sure that the 'PeopleService' webservice has been deployed ('`http://localhost:8080/sree/PeopleService`').
- Verify that the *peopleservice-client.jar* file located in the *sree/WEB-INF/lib* folder is on the Style Studio classpath. (See *Configuring the Style Studio Classpath*.)

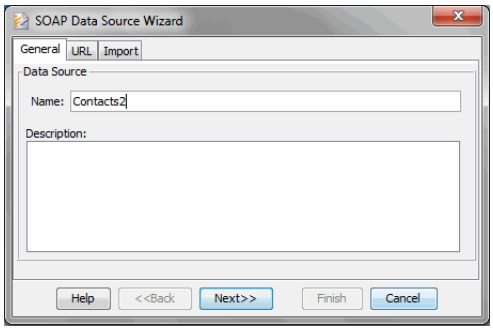
Then follow the steps below:



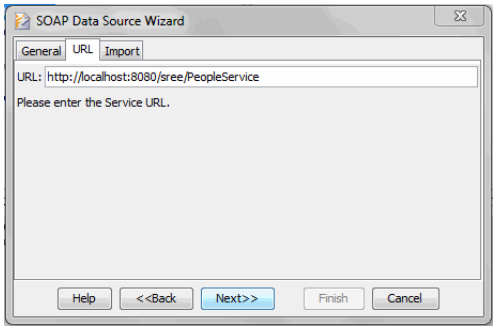
1. Press the 'New' button in the Style Studio toolbar. This opens the 'New Asset' dialog box.

2. In the left panel, select the 'Data Source' node. In the right 'Types' panel, select 'SOAP', and press 'OK'. This opens the 'SOAP Data Source Wizard'.

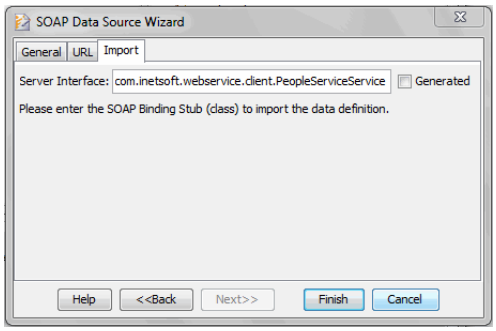
3. Select the **General** tab. In the 'Name' field, enter "Contacts2".



4. Select the **URL** tab. In the 'URL' field, enter the service URL: '`http://localhost:8080/sree/PeopleService`'

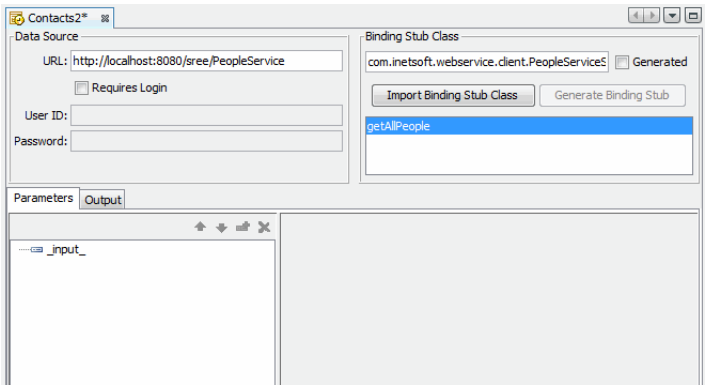


5. Select the **Import** tab. In the 'Server Interface' field, specify the client stub class (Soap Binding Stub): '`com.inetsoft.webservice.client.PeopleServiceService`'



Note: The 'Generated' option can be used to generate the client stub automatically if the web service does not require special intervention for the generation of the client (i.e., if a working client can be generated by `wsimport` with no arguments or custom binding).

6. Press 'Finish' to add the data source to the data source registry (*datasource.xml*).





7. Press the 'Test Data Source' button on the Style Studio toolbar to verify that the data source is configured correctly.

8. (Optional) Press the 'Import Binding Stub Class' button to import (introspect) its 'get' methods.

You can now create a query based on the 'get' methods of the data service.

4.4.2 Creating the Client Stub

You can create a client stub using any standard SOAP implementation that conforms to the JAXWS standard. The recommended implementation is Metro, developed by the GlassFish group.

To create a Java client stub for a web service using Metro, follow the steps below:

1. Make sure the web service is deployed and running on the host server.

2. Create the client java stubs:

   a. Download and set-up Metro (http://metro.java.net).

   b. Create a new folder where the stub files will be generated (e.g., `{StubHome}`).

   c. Create two folders in `{StubHome}`, e.g., *classes* and *source*, or any other suitable names.

   d. Use the *{MetroHome}/bin/wsimport* utility to create the stubs with the following command:

   ```
   wsimport.bat -s {folder1} -c {folder2} -Xendorsed
   {URL}?wsdl

   e.g., change directory to {StubHome}
   {MetroHome}/bin/wsimport.bat -s source -c classes -
   Xendorsed http://localhost:8080/sree/PeopleService?wsdl
   ```

   This will create the Java stub files in the classes folder and source code (optional) in the source folder.

4.5 Text Data Source

A Text data source is used to represent data that is stored in a fixed-width text file, delimited text file, or Excel spreadsheet. The text file is treated as a table: Each line is converted into a table row, and the fields are used to populate the columns.

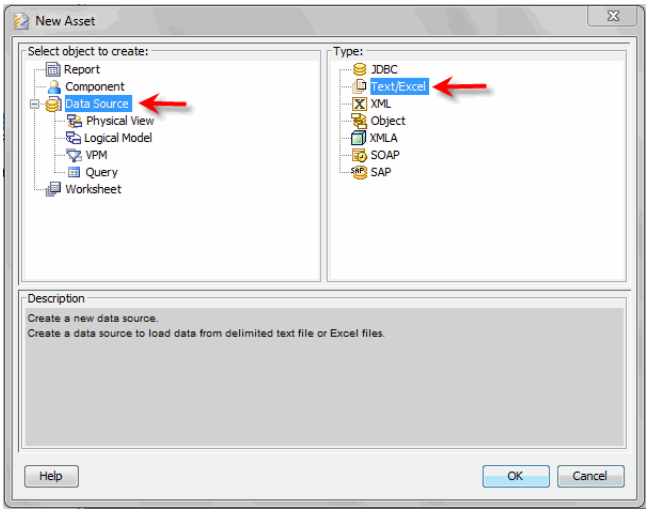
4.5.1 Defining a Text Data Source

A Text data source definition contains configuration information for accessing the data in the text file. The configuration includes the URL of the data source and a list of related requests. In a servlet- or CGI-based URL, each request corresponds to a parameter set for the HTTP request.

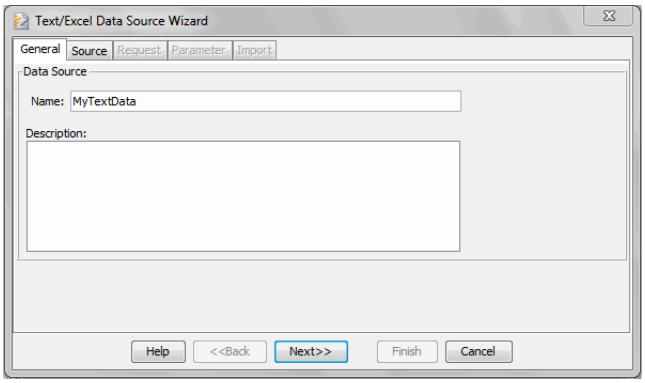
To create a Text data source, follow the steps below:

1. Press the ‘New’ button in the Style Studio toolbar. This opens the ‘New Asset’ dialog box.

2. In the left panel, select the ‘Data Source’ node. In the right ‘Types’ panel, select ‘Text’, and press ‘OK’. This opens the ‘Text Data Source Wizard’.



3. Select the **General** tab. In the 'Name' field, enter a name for the new data source.



4. Select the **Source** tab. In the 'URL or File' field, enter the path to the text data file.

   You can use a variable in the 'URL or File' field, e.g., '`file:///c:/datafiles/$(parameter_name)`' to make the data source URL dynamic (in part or whole). Parameters can be specified as system variables, as report parameters within the *sree.properties* file, or as URL parameters. The report engine will look for the parameters in the above sequence, and, if not found, will default to prompting the user for the parameters.

```
If the URL refers to a static data stream, simply create one
default request with no parameters.
```

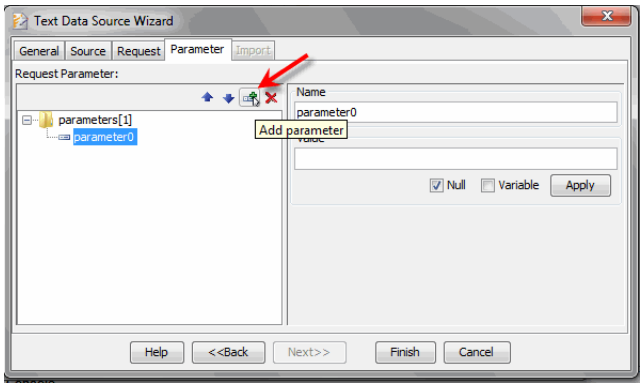
5. (Optional) Select the **Request** tab, and enter the name of the request. In a servlet- or CGI-based URL, each request corresponds to a parameter set for the HTTP request.

6. Add the desired parameters by following the steps below:

   a. Select the **Parameter** tab.

   b. Press the ‘Add parameter’ button, and specify the ‘Name’ and ‘Value’ of the parameter.

Click ‘Remove Parameter’ to delete an existing parameter.

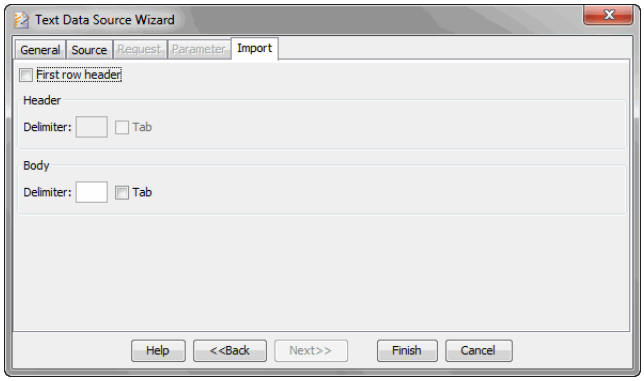


   c. To specify a null value for the parameter, select the ‘Null’ option.

   d. To specify a variable name for the parameter, select the ‘Variable’ option and click ‘Apply’. (When the ‘Variable’ option is selected, the value of the parameter is expected to be provided by an external source, such as report script or user input.)

   e. Repeat the above steps to add additional parameters.

7. (Optional) For a text or CSV file, select the **Import** tab, and make the following settings:



   a. If the first row contains header information, enable ‘First row header’.

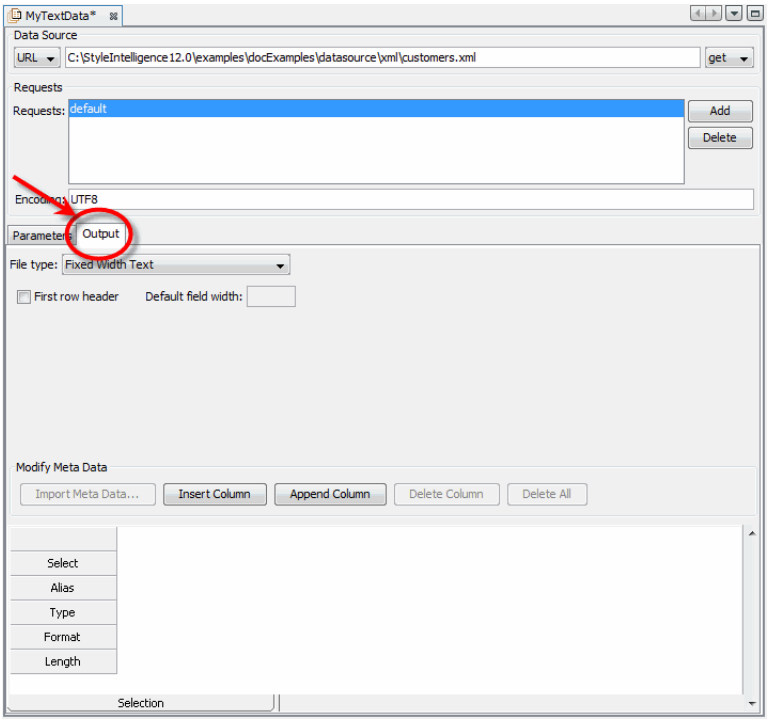
b. In the ‘Header’ panel, enter the character used as the header delimiter, or select ‘Tab’ for a tab-delimited header.

c. In the ‘Body’ panel, enter the character used as the body delimiter, or select ‘Tab’ for a tab-delimited body.

8. Press ‘Finish’ to close the Wizard. Style Studio displays the completed Text data source.

Note: If you already entered information on the **Import** tab of the Wizard, you do not need to complete the steps below.

9. Select the **Output** tab.



10. In the ‘File type’ menu, select the type of text file.

‘Fixed Width Text’ is appropriate for a file in which each row in a given column occupies the same width. (Different columns may have different widths.) ‘Delimited Text’ is appropriate for a file in which each column is separated by a space, comma, or other delimiting character. The two ‘Microsoft Excel’ options can be used for spreadsheets of the respective types.

11. If you selected ‘Fixed Width Text’, follow the steps below:

a. Select ‘First header row’ if the first row of the text file contains headers describing the columns (meta-data).

b. If you selected ‘First header row’, specify the number of characters in the column headers using the ‘Default field width’ box. This also sets the default width for the data rows, which you can modify later on a column-by-column basis.

12. If you selected ‘Delimited Text’, follow the steps below:

    a. Select ‘First header row’ if the first row of the text file contains headers describing the columns (meta-data).

    b. If you selected ‘First header row’, enter the delimiter character in the ‘Delimiter’ field of the ‘Header’ region. If the delimiter is a ‘Tab’ character, check the ‘Tab’ box.

    c. Enter the delimiter character in the ‘Delimiter’ field of the ‘Body’ region. If the delimiter is a ‘Tab’ character, check the ‘Tab’ box.

The default delimiter is a comma.

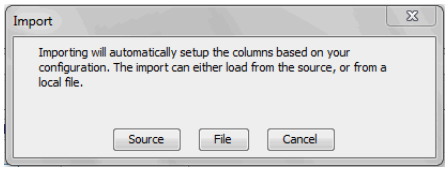
13. If you selected one of the ‘Microsoft’ formats, follow the steps below:

    a. Select the desired Excel sheet from the ‘Sheet’ menu.

    b. Select ‘First header row’ if the first row of the file contains headers describing the columns (meta-data).

    c. Enter the starting and ending rows and columns in the appropriate fields.

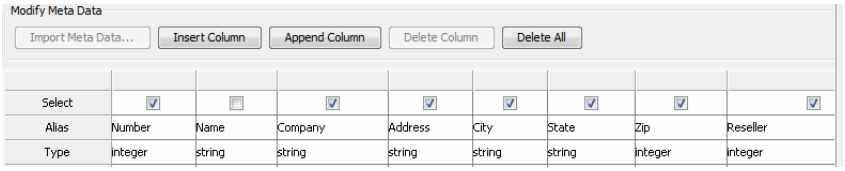
14. If you selected the ‘First header row’ option for any of the available formats, press the ‘Import’ button to import the column header meta-data. This opens the ‘Import’ dialog box.



The ‘File’ option is useful when you are using a ‘Server’ type repository and the datasource file path is not accessible locally.

15. Press ‘Source’ to load the meta-data from the specified Text data source. Press ‘File’ to load the meta-data from an alternate file that you select.

    The ‘Import’ feature uses the column headers and column data to populate the ‘Alias’ and ‘Type’ fields in the meta-data table at the bottom of the screen.



To make manual adjustments to the meta-data assignments, see *Modifying Text Source Meta-Data*.

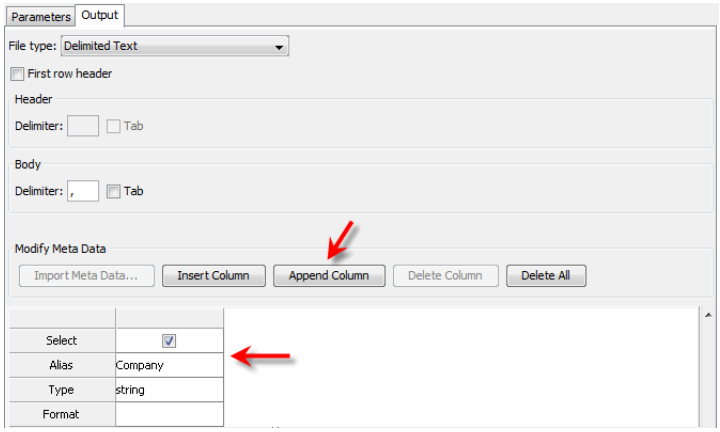
16. Press the 'Save' button in the Style Studio toolbar to save the new Text data source.

4.5.2 Modifying Text Source Meta-Data

If you do not wish to use the 'Import' function to automatically load the meta-data from the file (or if the file does not contain header information), you can define the data columns manually. Follow the steps below:

1. Press 'Append Column' in the bottom panel to add a column to the table specification.

2. In the 'Alias' field for the new column, enter the desired column header text.



3. In the 'Type' menu, select the desired data type for the column.

4. In the 'Format' field, select the desired format for the column. The format specifications are the same as those for the XML data source. See *Defining an XML Data Source* for more details.

5. Repeat the above steps to add additional column specifications. (To add a new column to the *left* of an existing column, use the 'Insert Column' button.)



6. Press the ‘Save’ button in the Style Studio toolbar to save the data source.

The example below illustrates how to specify multiple columns.

Example: Text Data Source

In this example, you will create a Text data source using a sample text file in the *examples/docExamples/datasource/xml* directory. Follow the steps below:

1. Press the ‘New’ button in the Style Studio toolbar. This opens the ‘New Asset’ dialog box.

2. In the left panel, select the ‘Data Source’ node. In the right ‘Types’ panel, select ‘Text’, and press ‘OK’. This opens the ‘Text Data Source Wizard’.

3. Select the **General** tab. In the ‘Name’ field, enter the data source name. For this example, enter “Text”.

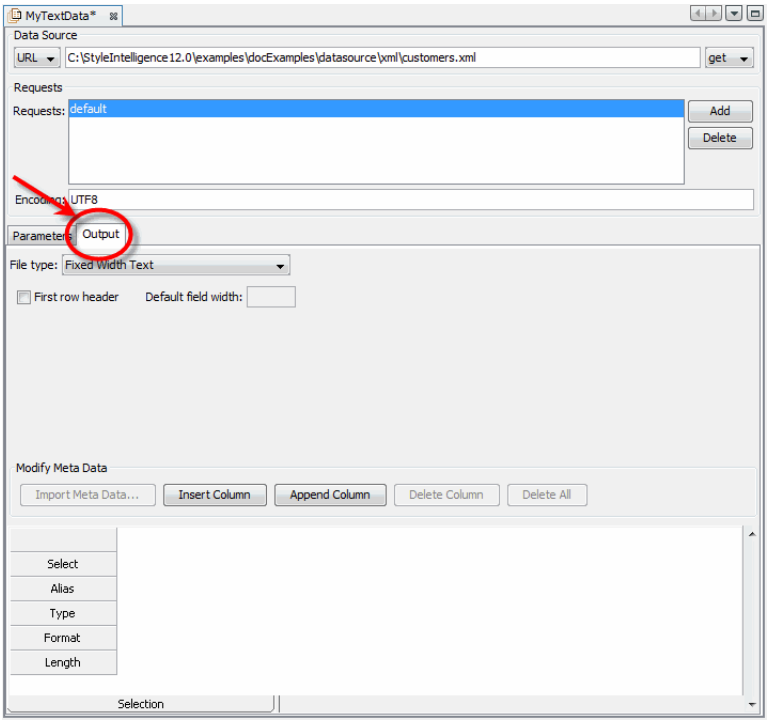
4. Select the **Source** tab. Press the ‘Browse’ button next to the ‘URL or File’ field, and select the following text file:

   ```
   {StyleIntelligence}\examples\docExamples\datasource\xml\customers.txt
   ```

Because this is a static text file, no parameters are needed.

5. Press ‘Next’ twice, and then press ‘Finish’ to close the Wizard. Style Studio displays the completed Text data source.

6. Select the **Output** tab.



7. In the ‘File type’ menu, select ‘Delimited Text’.

8. Deselect ‘First row header’, because the *customers.txt* file does not include a header row.

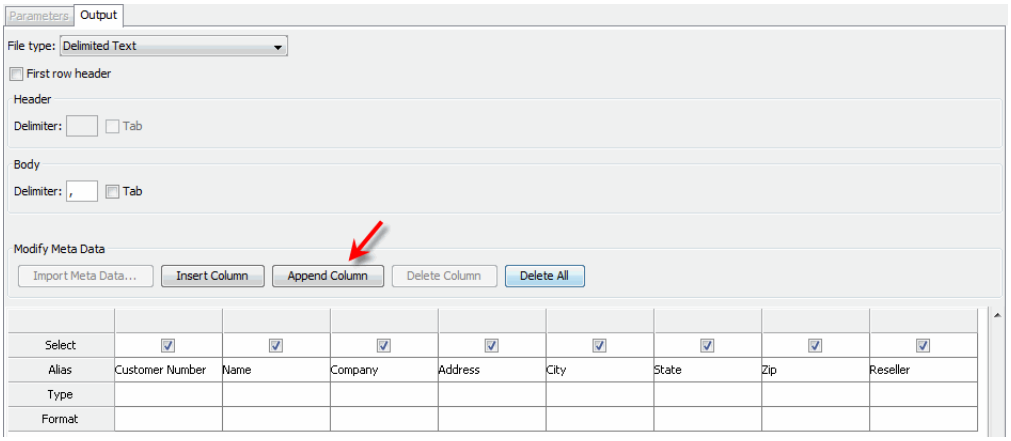
9. For the Body ‘Delimiter’, use the default comma delimiter.

For files with a header row, use the ‘Import’ feature to help define columns. See *Defining a Text Data Source*.

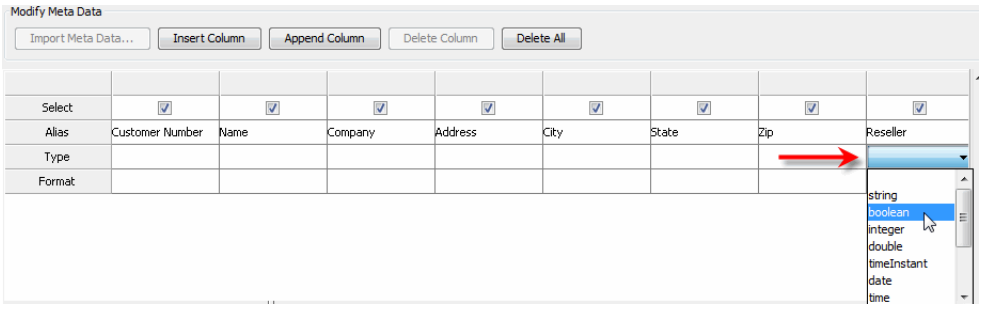
10. Press ‘Append Column’ to add a column to the table specification.

11. In the ‘Alias’ field for the new column, type “Customer Number ”.

12. Repeat the two previous steps to add the rest of the columns in the *customers.txt* file: “Name”, Company”, “Address”, “City”, “State”, “Zip”, and “Reseller”.



13. To set a column's data type, click in the correspoding 'Type' field, and make a selection. In this example, change the 'Reseller' column type to '`boolean`'.



14. Press the 'Save' button in the Style Studio toolbar to save the data source.

¢

4.6 Java Object Data Source

A session EJB can be used as a Java Object data source.

A Java Object Data Source is useful when your data cannot be accessed via common standards such as JDBC, SOAP, XML, etc., or if you have a pre-existing JAVA API to which you must connect.

To create a Java Object data source you must create a *Data Loader* class. A Data Loader class can be based on the *inetsoft.uql.object.DataLoader* interface, the *inetsoft.uql.object.DataLoader2* interface, or on *introspection* (by complying with JavaBean conventions). In some cases when using the *DataLoader* interface, an additional *Data Helper* class is also needed to describe the structure of the data being returned.

4.6.1 Selecting the Type of Data Loader

If your data has a flat tabular structure, the Data Loader should be based on the *DataLoader2* interface. This is the simplest and most convenient

option, as it deals only with primitive data types and does not require a Data Helper class.

For data that has hierarchical structure (especially recursive hierarchy) the Data Loader should be based on the *DataLoader* interface or on introspection.

The following are common examples of hierarchical structure:

- One forest has N trees, one tree has M leaves.
- An ‘Employee’ has a ‘Manager’ who is also an ‘Employee’, etc.

Data Loader based on Introspection

A Data Loader that does not implement the *DataLoader* interface is therefore completely based on introspection and must satisfy the following requirements:

- The class must have a default constructor that requires no arguments.
- The class must have methods that return a collection of objects. The collection may be an object of one of these subclasses:
  - `java.util.Collection`
  - `java.util.Enumeration`
  - `java.util.Iterator`

When a data source is defined based on the Data Loader, the query engine uses introspection to discover all public methods that return one of the collection types. Each method is then added to the data source as a request.

If a request is created from a method that requires parameters, you can specify parameter values as part of the data source definition or declare them as variables, which then become the data source parameters.

For each request, you need to specify the object type (class) returned by the request. The query engine probes the class to find all of the public “getter” methods, and then treats these as the fields in the record. During probing the query engine follows the object references, so the meta data can be a hierarchical structure, as shown below:

```
Employee (object class)
   Name
   FirstName
   LastName
   Address
      City
      State
      Zip
```

Data Loader based on DataLoader Interface

While the default introspection-based Data Loader works well with the classes that match the defined convention, the *DataLoader* interface provides a more flexible way to define the procedure to load objects. The design and runtime of the data source works in the same way as an introspection-based Data Loader, except that the *DataLoader* interface is used for meta-data discovery and runtime execution without relying on introspection.

The `DataLoader` interface defines the following four methods:

- **public String[] getRequests()**
  This method should return a list of request names. This corresponds to the methods that return a collection of objects in the introspection-based Data Loader.
- **public Class getRequestOutput(String request)**
  This method returns the object type for each request. It must handle all request names returned from `getRequests()`.
- **public Class getRequestParameter(String request)**
  This method returns the parameters for each request, similar to the `getRequestOutput()` method.
- **public Collection execute(String request, Object params, XNodePath condition)**
  This method executes the requests. It optionally handles the user defined conditions in the `XNodePath`. If any conditions defined in the `XNodePath` can be handled by the `execute()` method, they should be handled internally (for example, as a condition to an object oriented database query). If a condition cannot be handled by the `execute()` method, it should raise a `ConditionNotSupportedException`. The query engine will call the `execute()` method again without the condition, and will process the conditions as part of the post-processing.

Data Loader based on DataLoader2 interface

The *DataLoader2* interface is used to extract data that has a flat tabular structure, as opposed to data with an inherent hierarchy. This is the simplest way to extract tabular data. It requires no Data Helper class because it deals only with primitive data types.

The *DataLoader2* interface defines the following four methods:

- **public String[] getRequests()**
  This method should return a list of request names. A request is tied to one set of output objects sharing the same class, and to one set of parameters. Requests are presented in Style Studio and are selected as part of a query, e.g:

```
public String[] getRequests() {
  return new String[]
  {"salesForEmployee", "salesForState"};
}
```

- **public ObjectMetaData getRequestOutput(String request)**
  This method gets the output object type of a request. This method must support all requests returned from the `getRequests()` method, e.g:

```
public ObjectMetaData getRequestOutput(String request) {
  if(request.equals(“salesForEmployee“)) {
    return new ObjectMetaData(
      new String[] {"Employee", "Sales", "Year"},
      new Class[] {
        (new String()).getClass(),
        (new Float(0)).getClass(),
        (new Integer(0)).getClass()
    });
  }
  else if(request.equals(“salesForState“)){
    ....
  }
}
```

- **public ObjectMetaData getRequestParameter(String request)**
  This method gets the request parameter type. The parameter can be constructed when building a query and is passed to the data loaded in `execute()`.

```
public ObjectMetaData getRequestParameter(String request) {
  if(request.equals(“salesForEmployee“)) {
    return new ObjectMetaData(
      new String[] {"fiscalYear"},
      new Class[] {(new Integer(0)).getClass()});
  }
  else if(...) {
    ...
  }
}
```

- **public Object execute(String request, VariableTable params, XSelection columns, XNodePath condition) throws ConditionNotSupportedException**
  This method executes a request. If any condition is defined, the condition is passed to the `execute()` method. The method may choose to handle the conditions, or throw a `ConditionNotSupportedException`. If the exception is thrown, the engine will call `execute()` again without the condition parameter passed in, and will handle the filtering as part of the post processing. The following return types are supported: `XTable` object, `XTableNode` object, or a two-dimensional array in which the first row serves as the column headers.

Using a Data Helper

The Data Helper may be required under the following circumstances:

Your Data Loader is based on the *DataLoader* interface or introspection.

- **AND** the return object class has only primitive attributes, but does not follow the Java Bean naming conventions.
- **OR** the return object class has a hierarchy.

The Data Helper class must implement the *inetsoft.uql.object.DataHelper* interface, which defines the following two methods:

- **public XTypeNode[] getAttributes(Class cls)**
  Given a class, this method should return an array of field types. Each type can be a primitive type such as those below, or a more complex type that represents an object tree:

```
– string (inetsoft.uql.schema.StringType)
– integer (inetsoft.uql.schema.IntegerType)
– date (inetsoft.uql.schema.DateType)
– etc.
```

- **public Object getValue(Object obj, String name)**
  This method returns the value of a field. The name is the name of the field, and the `obj` parameter is the data object.

The Data Helper overrides the introspection in two ways.

- At design time, introspection is not used to discover object fields. Instead, the `getAttributes()` method is called on the Data Helper to find the fields.
- At runtime, field values are returned using the `getValue()` method of the Data Helper, instead of calling the "getter" method on the data object itself.

4.6.2 Leveraging JavaBean Properties

If the custom Data Loader includes Java Bean properties (those with get/set methods), you can configure these properties in the data source definition. This allows you to write a more generic, reusable, data loader implementation. For example, if the custom data loader obtains SharePoint lists, and requires a URL, user name, and password, you can create an individual data source for each property combination (using the same data loader class), rather than writing multiple classes.

If a *BeanInfo* class is associated with the Data Loader, you can provide display names, descriptions, custom editors, etc., for the class. You can also define a "secret" attribute for passwords, etc., that will use a password field (masked text) in the interface and encrypt the value in the data source file.

4.6.3 Defining a Java Object Data Source

To define an Object data source, first place all necessary files into a single directory (e.g., a directory named '`object`'). The list of necessary files includes the following:

- Data Loader class
- Data Helper class, if any
- BeanInfo class, if any
- All the object classes

The parent directory of this '`object`' directory must be included in the classpath.

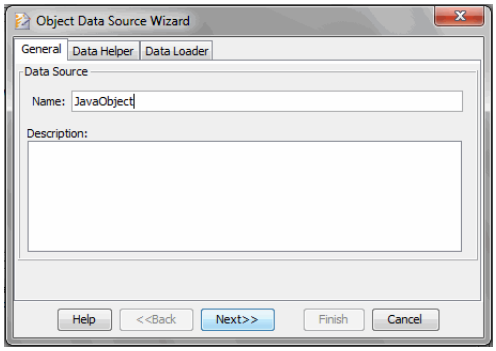
The example below creates an Object data source using a Data Helper and a Data Loader based on introspection. The required files can be found under the *examples/docExamples/datasource/object* directory.

Example: Java Object Data Source

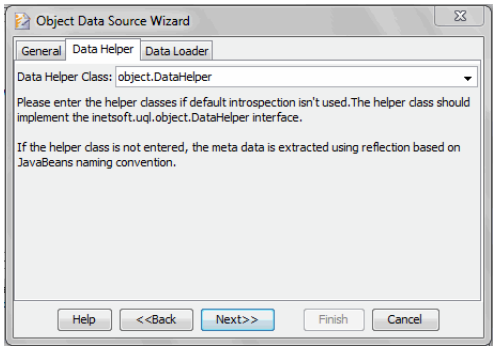
1. Press the 'New' button in the Style Studio toolbar. This opens the 'New Asset' dialog box.

2. In the left panel, select 'Data Source'. In the right 'Types' panel, select 'Object', and press 'OK'. This opens the 'Object Data Source Wizard' dialog box.

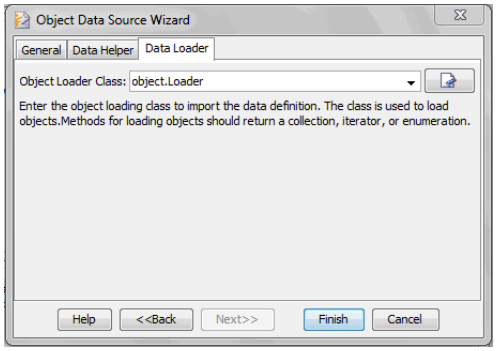
3. Select the **General** tab. In the 'Name' field, enter a name for the data source name, for example: "JavaObject".



4. Select the **Data Helper** tab. In the 'Data Helper Class' field, type in '`object.DataHelper`'.

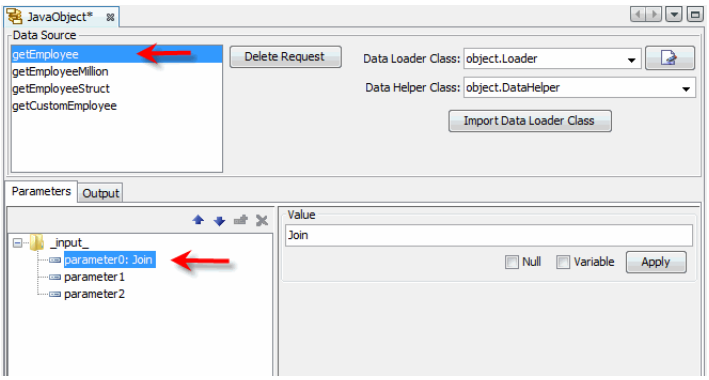


5. Select the **Data Loader** tab. In the 'Object Loader Class' field, type in '`object.Loader`'.



6. Press 'Finish' to create the data source.

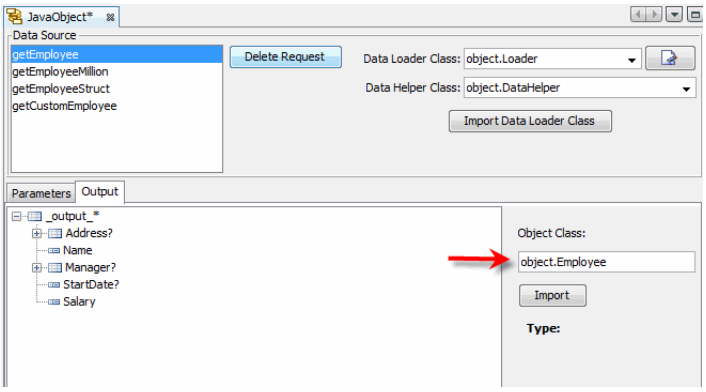
7. In the 'Data Source' panel, select the 'getEmployee' request. Select `parameter0`, and enter "John" in the 'Value' field. Press 'Apply'.



8. Select `parameter1`, and enter "5000" in the 'Value' field. Press 'Apply'.

9. Select `parameter2`. In the 'Value' field, pick any date and time. Press 'Apply'.

10. Select the **Output** tab. Type '`object.Employee`' in the right-side text field and press 'Import'.



11. In the 'Data Source' panel, select the 'getEmployeeStruct' request.

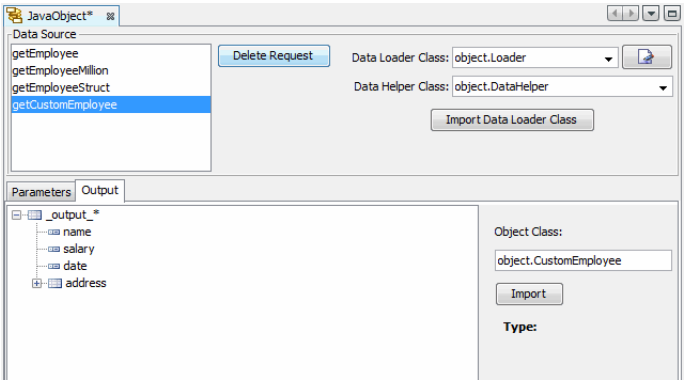
12. Select the **Output** tab, enter '`object.Employee`' in the right-side text field, and press 'Import Object Class'.

13. In the 'Data Source' panel, select the 'getEmployeeMillion' request.

14. Select the **Output** tab, enter '`object.Employee`' in the right-side text field, and press 'Import'.

15. In the 'Data Source' panel, select the 'getCustomEmployee' request.

16. Select the **Output** tab, enter '`object.CustomEmployee`' in the right-side text field, and press 'Import'.





17. Press the 'Save' button in the Style Studio toolbar to save the data source settings.

¢

The next example creates an Object data source using a custom data loader that implements the `DataLoader` interface. The required files can be found under the *examples/docExamples/datasource/object* directory.

Example: Custom Data Loader

1. Press the 'New' button in the Style Studio toolbar. This opens the 'New Asset' dialog box.

2. In the left panel, select 'Data Source'. In the right 'Types' panel, select 'Object', and press 'OK'. This opens the 'Object Data Source Wizard' dialog box.

3. Select the **General** tab. In the 'Name' field, enter the name for the data source name: "custom".

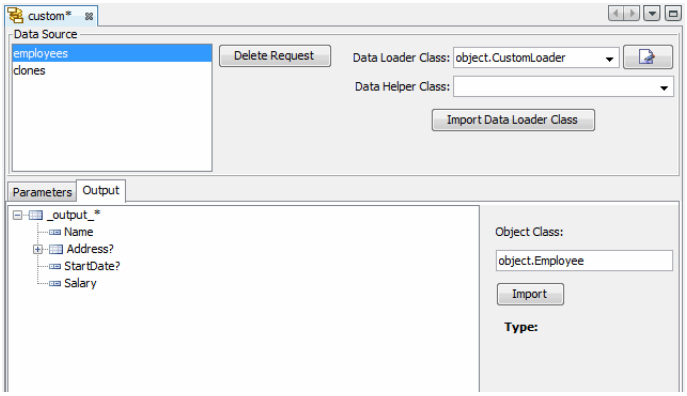
If you have declared a class using the standard Java loader mechanism, you can select the class name from the 'Data Helper Class' and 'Object Loader Class' menus.

4. Select the **Data Loader** tab. In the 'Object Loader Class' field, select or type '`object.CustomLoader`',

5. (Optional) If the custom data loader defines JavaBean properties, press the 'Properties' button next to the 'Object Loader Class' field, and enter values for these properties.

6. Press 'Finish'.

7. In the 'Data Source' panel, select the 'employees' request. Select the **Output** tab.



Note that the output object classes are automatically imported for both of the requests.



8. Press the 'Save' button in the Style Studio toolbar to save the data source settings.

¢

See Also

Querying a Java Object Data Source, for information on constructing a query.

4.7 SAP Data Source

To connect to an SAP data source, you must first instal the SAP Java Connector (JCo) drivers on your system. Follow the steps below:

1. Download and extract the JCo software to a new directory, e.g., *C:\SAPJCo*.

2. Copy the following files to the *C:\WINDOWS\SYSTEM32* directory:

   ```
   librfc32.dll
   sapjcorfc.dll
   ```

3. (Optional) If your Windows environment does not have the .NET SDK installed, copy the following .NET files to the *C:\WINDOWS\SYSTEM32* directory:

   ```
   msvcp71.dll
   msvcr71.dll
   ```

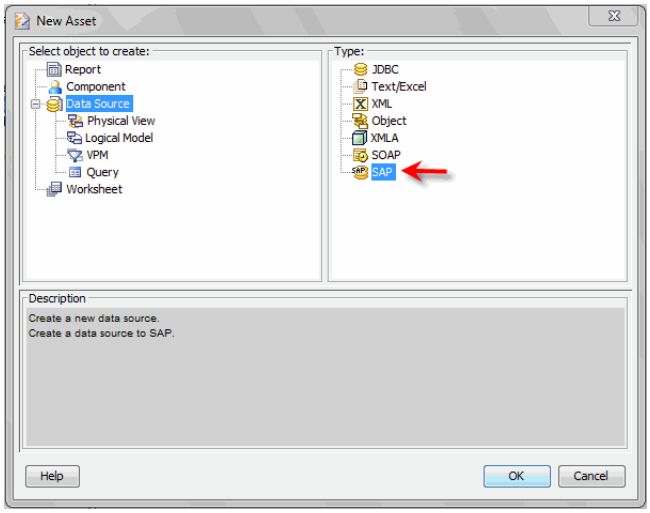
4. Add the *sapjco.jar* file to your Style Studio classpath. (See *Configuring the Style Studio Classpath*.)

5. Place a copy of the *sapjco.jar* file into the *webapps\sree\WEB-INF\lib* directory.

Once you have finished the above configuration, create the SAP data source by following the steps below:

1. Click the ‘New’ button in the Style Studio toolbar. This opens the ‘New Asset’ dialog box.

2. In the left panel, select the ‘Data Source’ node. In the right ‘Types’ panel, select ‘SAP’, and click ‘OK’.

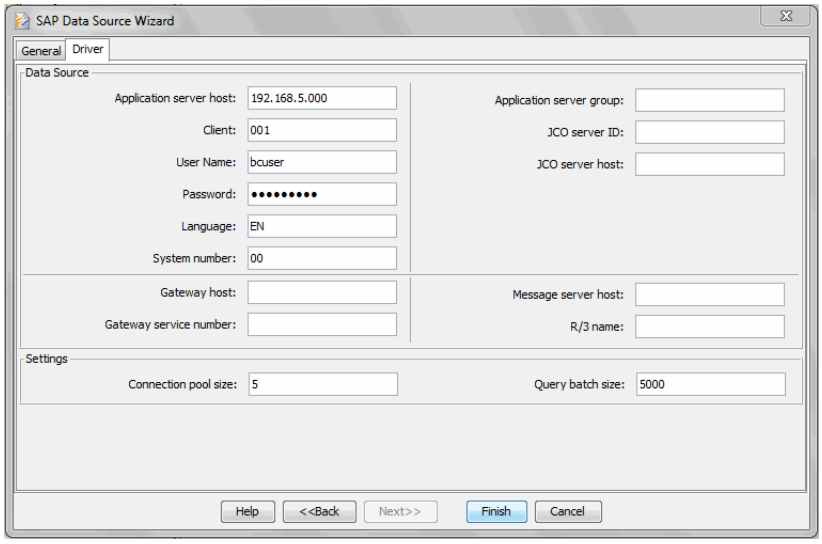


This opens the ‘SAP Data Source Wizard’.

3. Select the **General** tab. Enter a ‘Name’ and optional ‘Description’ for the data source.

4. Select the **Driver** tab, and enter the requested connection information.

   The following values are required: ‘Application server host’, ‘Client’, ‘User name’, ‘Password’, and ‘System number.’ See the SAP documentation for further information about the connection parameters.

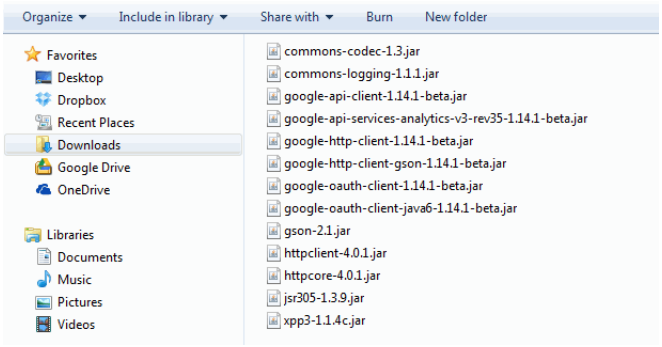


5. Press ‘Finish’ to close the Wizard. Style Studio displays the completed SAP data source.

4.8 Google Analytics Data Source

To connect to a Google Analytics data source, follow the steps below:

1. Download the *inetsoft-google-analytics.zip* file from the URL that was provided to you. This archive contains a file called *inetsoft-google-analytics-20130417.jar* (version number may differ) and a *lib* directory containing a set of JAR files such as those shown below.

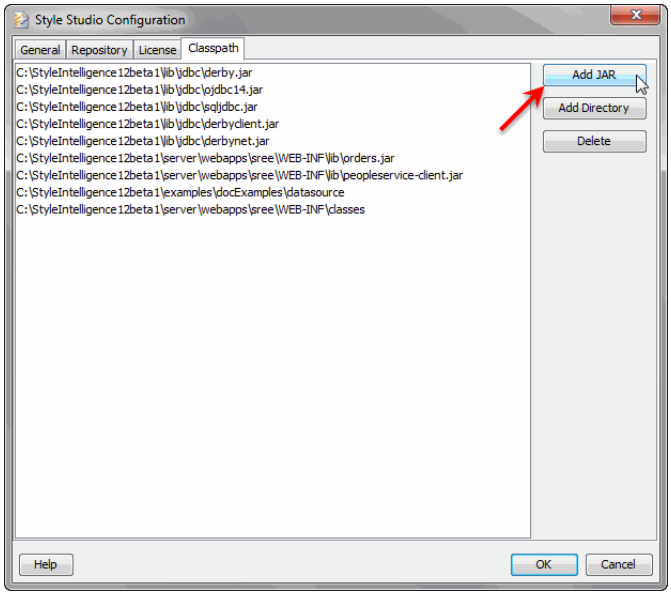


2. Extract all JAR files into the *sree/WEB-INF/lib* directory of the Style Intelligence server installation.

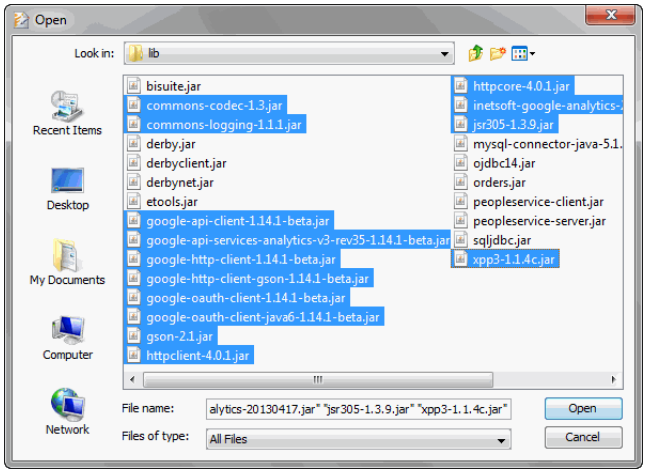
3. Launch Style Studio, and select 'Configure' from the 'File' menu. This opens the 'Style Studio Configuration' dialog box.

4. Select the **Repository** tab, and configure the working repository to point to *sree/WEB-INF/classes*. (See *Configuring a Repository* for more information.)

5. Select the **Classpath** tab, and press the 'Add JAR' button. (See *Configuring the Style Studio Classpath* for more information.)



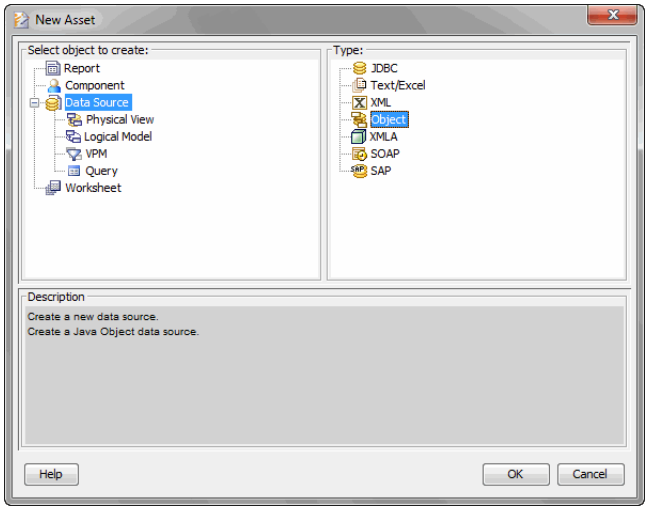
6. Navigate to the location where you copied the JAR files (*sree/WEB-INF/lib*). Select all of the extracted JAR files and press 'Open'. This adds the files to the classpath.



7. Press 'OK' to close the 'Style Studio Configuration' dialog box.

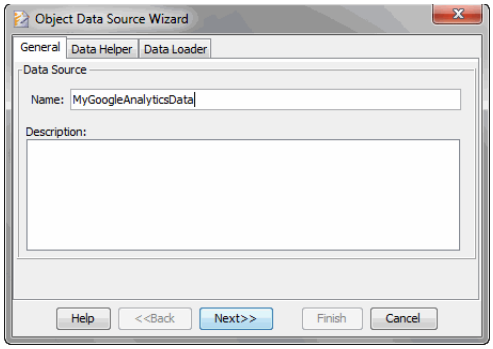
8. Select 'New' from the Style Studio 'File' menu. This opens the 'New Asset' dialog box.

9. In the left panel, select 'Data Source'. In the right panel, select 'Object'.



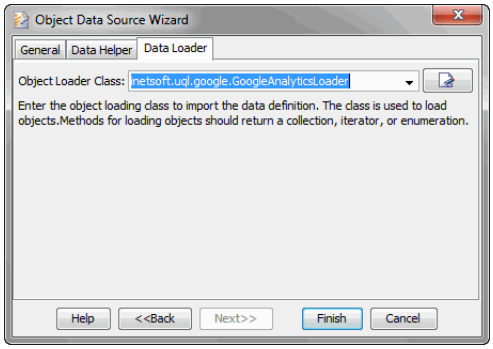
10. Press ‘OK’. This opens the Object Data Source Wizard.

11. Enter a ‘Name’ for the new data source.

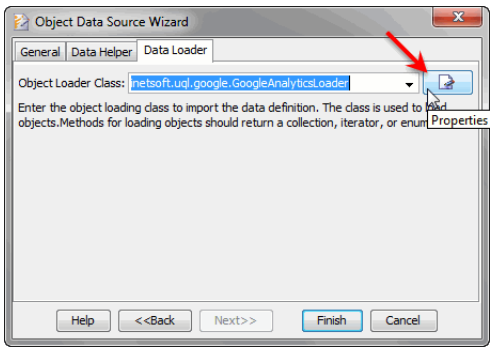


`Note: You do not need to provide any information in the` **`Data Helper`** `tab.`

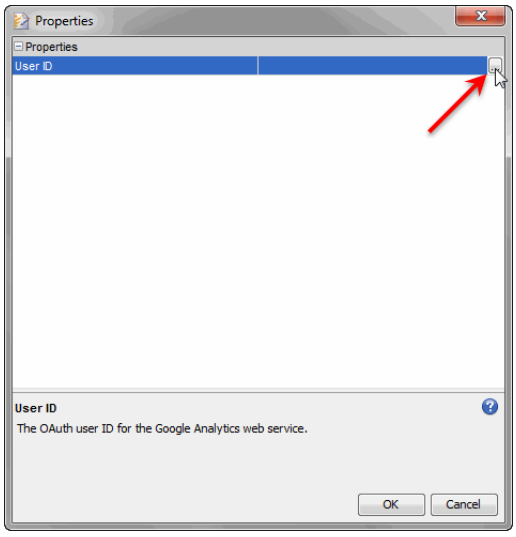
12. Click on the **Data Loader** tab. In the ‘Object Loader Class’ menu, select the `inetsoft.uql.google.GoogleAnalyticsLoader` class.



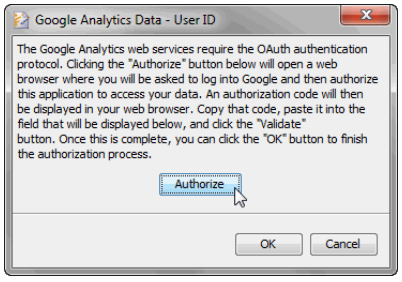
13. Press the ‘Properties’ button (next to the ‘Object Loader Class’ menu). This opens the ‘Properties’ dialog box.



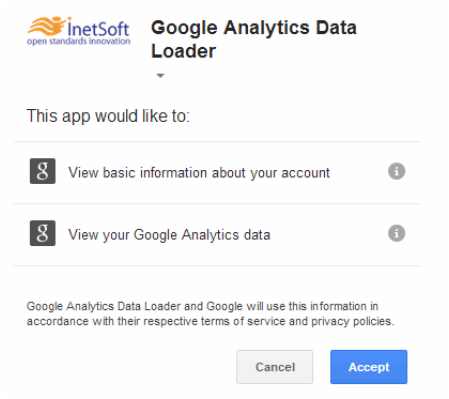
14. Press the ‘…’ button in the ‘Properties’ dialog box. This opens the ‘Google Analytics Data–User ID’ dialog box.



15. Press the ‘Authorize’ button. This opens a web page that may prompt you to log into Google (if you are not logged in already).

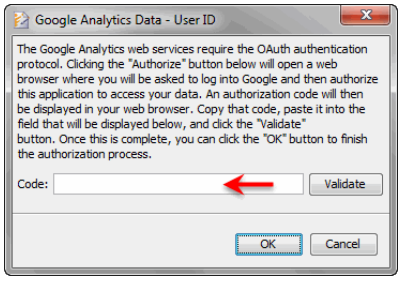


16. Log into the desired account. This prompts you to accept an agreement.



17. Press the ‘Accept’ button. This displays a code in the browser window.

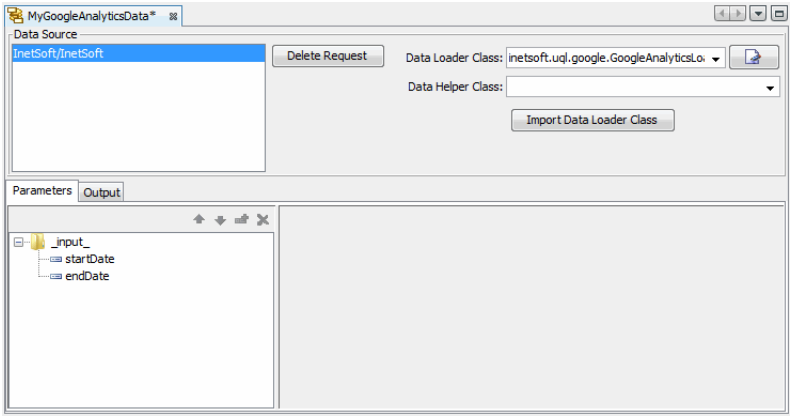
18. Copy the code from the browser window, and paste the code into the ‘Google Analytics Data–User ID’ dialog box. Press ‘Validate’. (Validation may take a few seconds.)



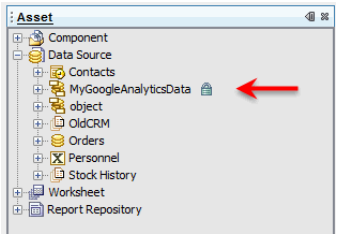
19. When the code has been validated (indicated by an “Authorization successful” message), press ‘OK’ to close the dialog box.

20. Press ‘OK’ to close the ‘Properties’ dialog box.

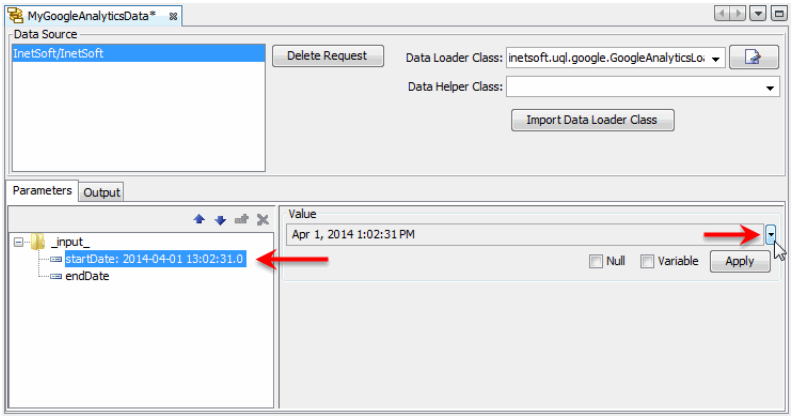
21. Press ‘Finish’ to close the ‘Object Data Source Wizard’. This opens the data source in Style Studio for editing.



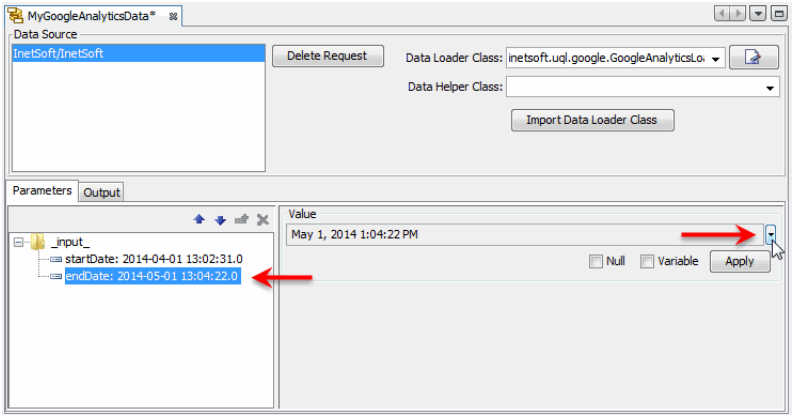
22. Press the 'Save' button in the Style Studio toolbar to save the data source. The saved data source now appears in the Style Studio Asset panel under the 'Data Source' node.



23. In the **Parameters** panel, click on the 'startDate' parameter. Click the arrow button on the right side of the date field, and enter a start date for the period of interest. Press 'Apply'.

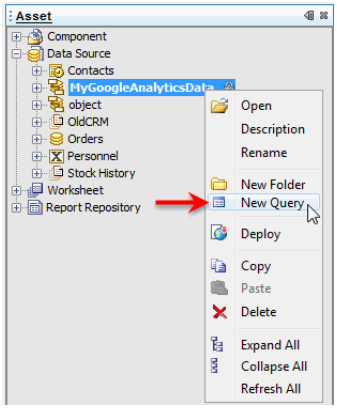


24. In the **Parameters** panel, click on the 'endDate' parameter. Click the arrow button on the right side of the date field, and enter an end date for the period of interest. Press 'Apply'.



25. Press the 'Save' button in the Style Studio toolbar to save the data source with the specified start and end dates.

26. To create a query based on the new data source, right-click the data source in the **Asset** panel, and select 'New Query' from the context menu.



27. Proceed to define your query as described in *Querying Non-relational Data Sources*.

4.9 SharePoint Data Source

To connect to a Microsoft SharePoint system, follow the steps below:

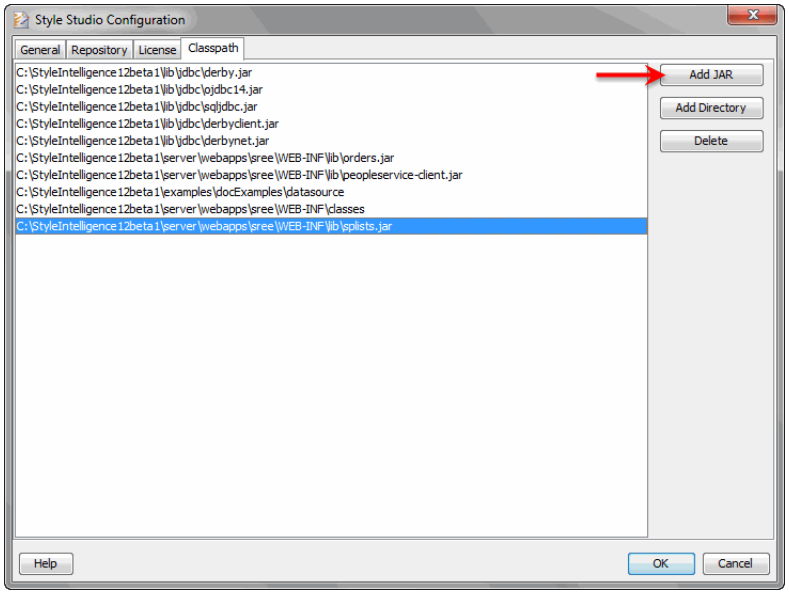
1. Download the *splists.jar* file from the URL that was provided to you.

2. Copy the *splists.jar* file into the *sree/WEB-INF/lib* directory of the Style Intelligence server installation.

3. Launch Style Studio, and select 'Configure' from the 'File' menu. This opens the 'Style Studio Configuration' dialog box.

4. Select the **Classpath** tab, and press the 'Add JAR' button. (See *Configuring the Style Studio Classpath* for more information.)

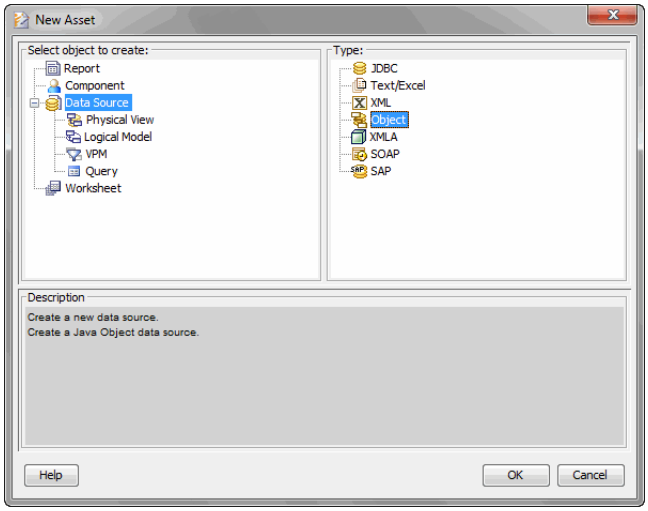
5. Navigate to the location where you copied the *splists.jar* file (*sree/WEB-INF/lib*). Select *splists.jar* and press 'Open'. This adds the file to the classpath.



6. Press 'OK' to close the 'Style Studio Configuration' dialog box.

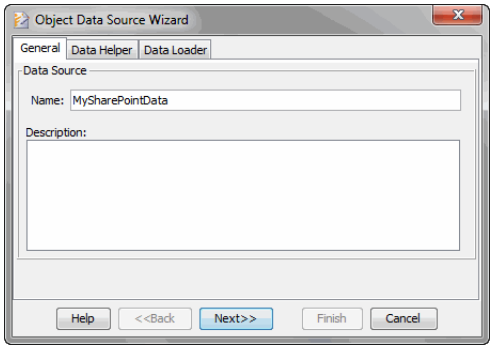
7. Select 'New' from the Style Studio 'File' menu. This opens the 'New Asset' dialog box.

8. In the left panel, select 'Data Source'. In the right panel, select 'Object'.



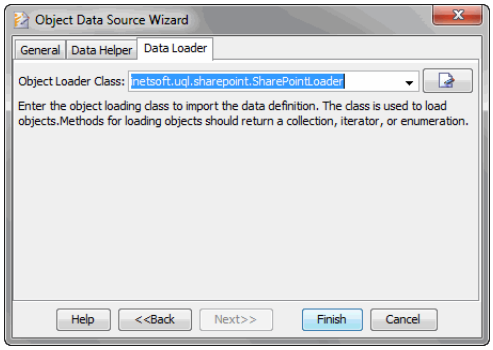
9. Press ‘OK’. This opens the Object Data Source Wizard.

10. Enter a ‘Name’ for the new data source.

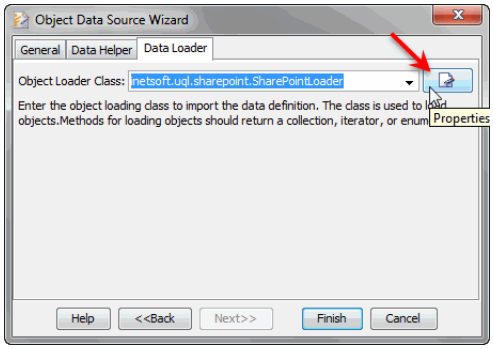


`Note: You do not need to provide any information in the` **`Data Helper`** `tab.`

11. Click on the **Data Loader** tab. In the ‘Object Loader Class’ menu, select the `inetsoft.uql.sharepont.SharepointLoader` class.



12. Press the ‘Properties’ button (next to the ‘Object Loader Class’ menu). This opens the ‘Properties’ dialog box.



13. From the ‘Authentication’ menu, choose the authentication method:

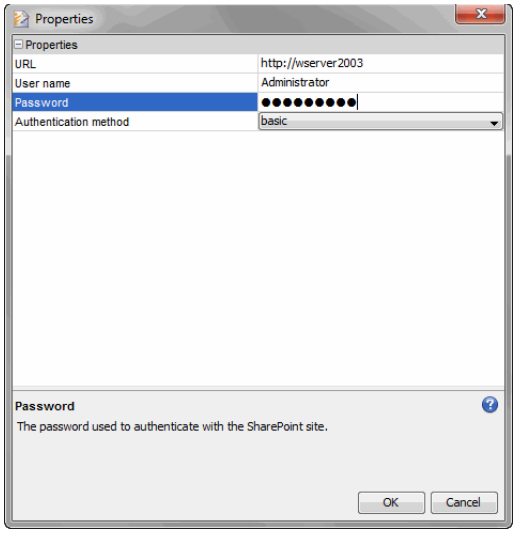
a. Select ‘basic’ for most cases.

b. Select ‘online’ to login to a hosted SharePoint system.

c. Select ‘windows’ to login with your Windows user account.

   If you are using Windows authentication, you will need a login module configuration file and Kerberos 5 configuration file. Specify the locations of these files using JVM arguments `Djava.security.auth.login.config` and `Djava.security.krb5.conf` for both the server and desktop application JVMs. See the appropriate GSS-API documentation for further details.

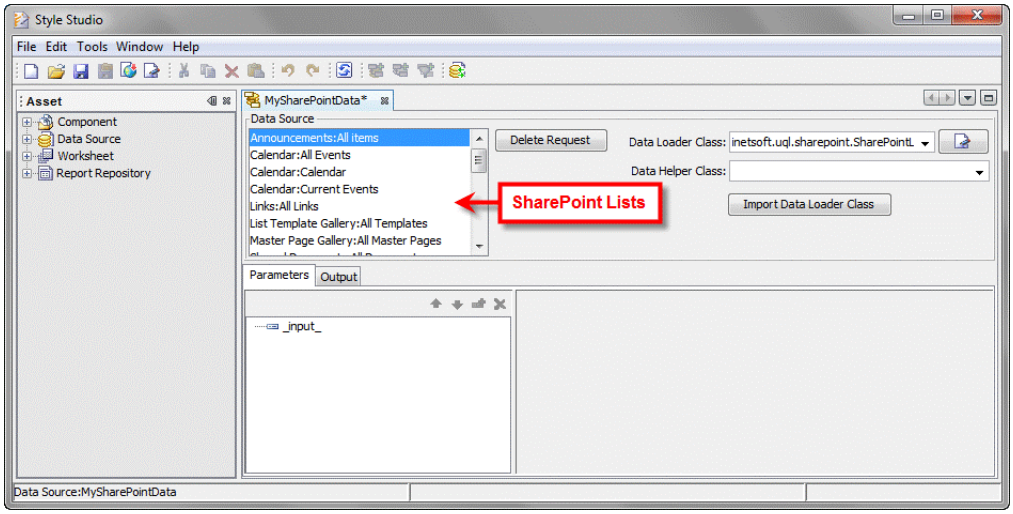
14. Enter your SharePoint server URL in the ‘URL’ field, and enter an appropriate user name and password in the ‘User Name’ and ‘Password’ fields.



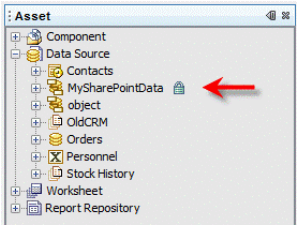
15. Press ‘OK’ to close the ‘Properties’ dialog box.

16. Press ‘Finish’ to close the ‘Object Data Source Wizard’.

    This opens the data source in Style Studio for editing. Observe that your SharePoint Lists are represented as individual requests within the ‘Data Source’ panel.



17. Press the 'Save' button in the Style Studio toolbar to save the data source. The saved data source now appears in the Style Studio **Asset** panel under the 'Data Source' node.



4.10 Querying Non-relational Data Sources

While queries on non-relational data sources do not use SQL, they can be used to filter and manipulate the data before returning the result set. This also provides a common access metaphor, because all queries are interchangeable.

4.10.1 Querying XML or Web Service Data Source

A Web Service query is very similar to an XML query. Therefore, the following example will explain how to create a query in either of the two data sources.

An XML query consists of three major components: XML sub-tree selection, tree node filtering, and table construction.

See Also

Web Service Data Source, for information on how to configure an XML data source.

XML Data Source, for information on how to configure an XML data source.

Selection Tree Path

Walkthrough

The first step in creating an XML query is to select a branch of the XML tree. The selection tree path is a required component in XML queries.

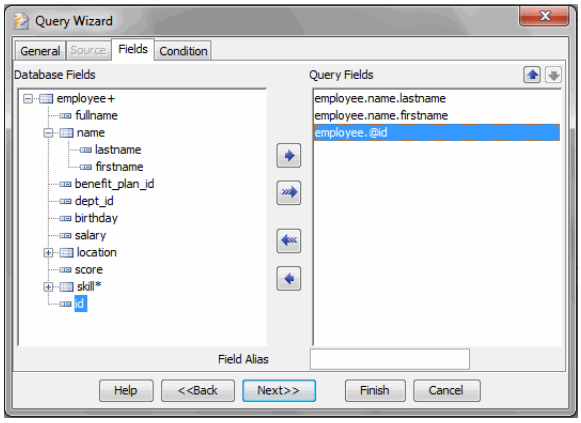
Follow these steps:

1. Click the ‘New’ button in the Style Studio toolbar. This opens the ‘New Asset’ dialog box.

2. In the left panel, select the ‘Query’ node. In the right ‘Types’ panel, select the ‘Personnel’ data source, and click ‘OK’. This opens the ‘Query Wizard’.

3. Select the **General** tab. In the ‘Name’ field, type “all_employees”.

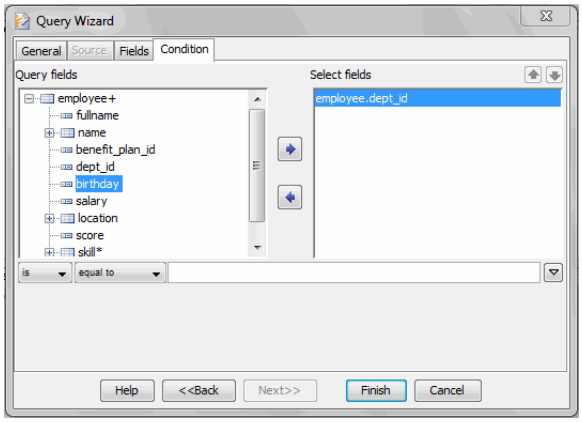
4. Select the **Fields** tab. Expand the ‘`name`’ node in the ‘Database Fields’ panel. Add ‘`lastname`’ and ‘`firstname`’ to the ‘Query Fields’ panel. Also add the ‘`id`’ node, then click ‘Finish’.



5. (Optional) Select the **Condition** tab. Click a node in the tree to filter, and press the ‘Add’ button.

6. Use the menus to specify the first part of a condition. For example, to select employees from a particular department, click to select the `dept_id` field, and enter the following condition fragment from the menus:

```
[employee.dept_id][is][equal to]
```



7. To enter a value in the condition, click the “arrow” button at the right side of the condition, and select the ‘Value’ option. Then enter the desired value in the text field.

8. To use a variable in the condition, click the “arrow” button at the right side of the condition, and select the ‘Variable’ option. In the adjacent ‘Name’ field, enter a name for the parameter. (For example, enter “`deptVar`” as the parameter name.)

   This is the name (case sensitive) by which you will refer to the parameter in hyperlinks, scheduled tasks, and scripts. If the condition operator is “`one of`”, then the variable represents an array. Otherwise, it is a scalar value.

Note that the adjacent menu automatically chooses the ‘equal to’ or ‘one of’ operator to match your selection.

9. To use session data in the condition, click the “arrow” button at the right side of the condition, and select the ‘Session Data’ option.

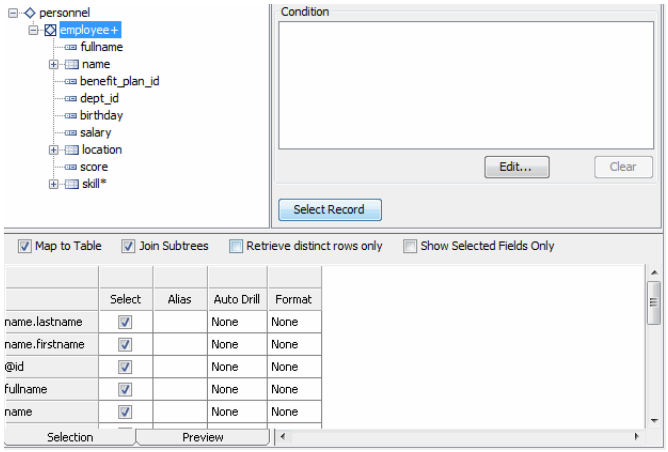
   In the menu, choose one of ‘User’, ‘Roles’, or ‘Groups’. These parameters return information about the user who is currently accessing the report; respectively, the user name, the array of roles to which the user belongs, and the array of groups to which the user belongs.

10. Repeat the above steps to add additional conditions.

11. Press ‘Finish’ to exit the Wizard.

12. Select the ‘`employee`’ node on the schema tree.

13. Click on ‘Select Record’ to select the sub-tree as the output. This marks the node as a selected path. See *Selection Nodes* for more information.



Selection Nodes



A tree path selects a branch of the output tree to serve as the root of the output data. Once a path is selected, the nodes on the path are changed to the path icon. A single diamond icon represents non-selection nodes, and a diamond icon in a box border represents selection nodes.

A node is a selection node if there can be multiple instances of the node in the tree. For example, there can be one or more ‘Employee’ elements under a ‘Personnel’ node. This is indicated by a plus sign at the end of the node label. Therefore, the ‘Employee’ node is a selection node. On the other hand, there can be only one ‘Personnel’ node in the XML stream, so this is a non-selection node.

Apart from differences in their visual appearance on the schema tree, a selection node can also have a query condition attached to it. If a query condition is associated with a selection node, the condition is used to select a subset of the nodes based on the condition evaluation. (See *Tree Node Filtering* for more information). For non-selection nodes there can be only one instance of the node, so selection is never necessary.

If the tree path contains multiple selection nodes and each selection node selects multiple instances of the nodes on that tree level, the tree path will select more than one subtree. For example, if you select the ‘`personnel.employee.skill`’ subtree, because there is more than one employee, the result of the tree path selection could contain multiple ‘Skill’ subtrees. In this case, the subtrees are merged into one root node, with all subtrees as its children.

Tree Transformation Rules

If a path in the schema tree is selected without any query conditions, the query will simply parse the XML output and return the selected subtrees as the result. The result tree is interpreted by the report engine based on the

binding of the query. If tabular data is expected by the element with which the query is associated, the tree is transformed into a table by using the following rules:

- If the tree contains multiple nodes of the same type, each node is converted to a table row. This is the case if any tree path node is a selection node.
- If the tree contains a single root node, this node is converted to a single table row. This is the case if the tree path does not contain any selection nodes.
- A sub-tree is converted to a table row by treating each child as a table column. The child node's name is used as the column name, and the child node's value is used as the column value.

Figure 2. Example tabular output

name.lastname	name.firstname	@id	fullname	name	benefit_plan_id	dept_id	birthday	salary
Chaney	Allen	10	Allen Chaney		2	4	1973-10-15	61000
Chaney	Allen	10	Allen Chaney		2	4	1973-10-15	61000
Natalevic	Anthony	14	Anthony Natalevic		3	5	1971-10-15	53000
Johnson	Ellen	1	Ellen Johnson		1	1	1966-10-15	68000
Johnson	Ellen	1	Ellen Johnson		1	1	1966-10-15	68000
Oakley	Frank	4	Frank Oakley		1	2	1954-9-6	79000
Brown	Gloria	11	Gloria Brown		3	4	1978-10-15	31000

Because the tree is converted to a table using the default table conversion, the child values of the name and location elements are not used in the final table. To display the values properly, you need to specify more precise table construction rules. See *Table Construction* for additional information.

Table Construction

You can specify table construction rules as part of a query. This gives you more control over how the tree is converted to a table. Additional conversion parameters are as follows:

- Inclusion of sub-tree children nodes as table columns.
- Inclusion of node attributes as table columns.
- Table column aliases.
- Data type conversion

Adding Table Construction Rules

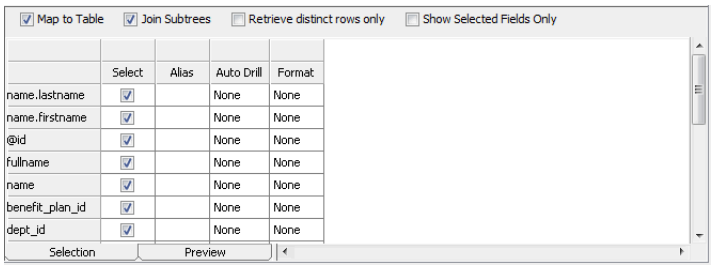
To add table construction rules to a query, follow the steps below:

1. Select the 'Map to Table' option. This enables the controls for specifying table construction rules.

2. (Optional) Select the ‘Join Subtrees’ option to create multiple rows per record, where each row corresponds to a particular combination of the child nodes of the subtrees.

   For example, if the selected node represents job application information, it might have subtree nodes for job location and job type. When ‘Join Subtrees’ is not selected, only one row is created for each application, representing the first child node for each subtree (e.g., ‘New York’ & ‘engineering’). If the ‘Join Subtrees’ option is enabled, multiple rows are created to pair each location with each job type (e.g., ‘New York’ & ‘engineering’, ‘Boston’ & ‘engineering’, ‘New York’ & ‘programming’, ‘Boston’ & ‘programming’, etc.). The values for other columns would be duplicated.

3. Change individual column attributes using the bottom selection table, where each available column is displayed. The column header shows the node or attribute name. The fields under the column name are column conversion attributes.



   a. Enable the ‘Select’ option for a column to include the column in the final output table.

   b. Enter a value in the ‘Alias’ field to use this value as the column header in the final table. Otherwise, the node or attribute name is used.

In the case of XML queries, you must perform data type conversion on the data source schema definition screen during data source definition. See *Defining an XML Data Source* for more information.

Table Construction Rules

Walkthrough

In this walkthrough, you will add some table construction rules to the ‘all_employees’ query from the example in *Selection Tree Path*. Follow the steps below:

1. Create a new query, ‘all_employees_table’, following the same steps from the previous example.

2. Enable the ‘Map to Table’ option.

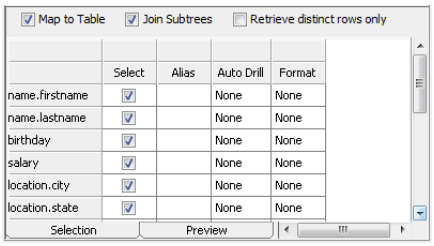
3. Deselect the ‘Join Subtrees’ option.

4. Uncheck the ‘Select’ field for the following columns:

   ‘fullname’
   ‘name’
   ‘benefit_plan_id’
   ‘dept_id’
   ‘location’
   ‘score’
   ‘skill’
   ‘@id’
   ‘skill.description’

5. Drag the ‘name.firstname’, ‘name.lastname’, ‘birthday’, ‘salary’, ‘location.city’, and ‘location.state’ columns to the front (left) of the table.

6. Enter the aliases for the columns.



The resulting query output now has the expected correct values.

Figure 3. Tabular Output With Joint Sub-Tree Selection

name.firstname	name.lastname	birthday	salary	location.s
Ellen	Johnson	1966-10-15	68000	East Bruns
Rebecca	Hartnack	1963-1-5	62000	East Bruns
Smita	Jacques	1972-4-15	68000	Middletowr
Frank	Oakley	1954-9-6	79000	Middletowr
Sherri	Shyko	1968-9-15	58000	New York
Robert	Lotzer	1969-8-15	65000	New York
Jeffrey	Jacob	1954-2-15	98000	New York

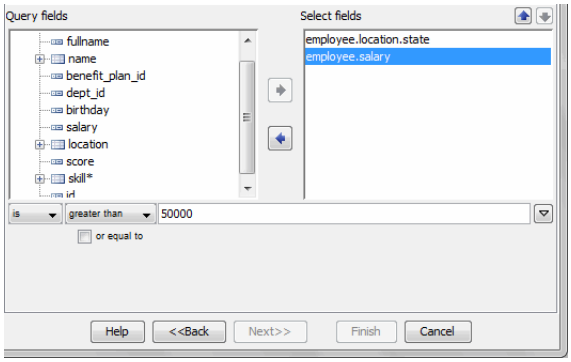
Tree Node Filtering

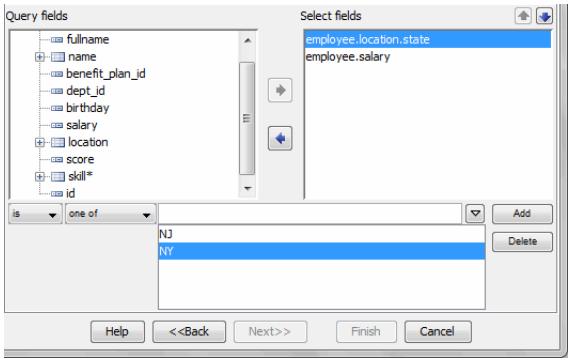
So far you have used the query engine to select all nodes on the specified tree path. As described before, you can attach conditions to selection nodes to filter the child sub-trees and extract a subset of the nodes.

Attaching a Conditional Expression

You can attach a conditional expression to each selection node on the tree path. The condition can be based on the values of any of the child nodes of the selection node. The expression syntax is based on the SQL conditions. For example, a condition attached to the ‘employee’ node of the ‘personnel’ tree can refer to the child of the ‘employee’ node by name as follows:

```
salary >= 50000 and location.state in (‘NY’, ‘NJ’)
```





When this condition is attached to the ‘employee’ node, the result of the query will only consist of the employees matching the criteria. The expression can refer to the immediate child of the selection node, or any descendent of the selection node. A child of the immediate child can be referenced by concatenating the name of the immediate child with the name of its child, separated by a dot. This notation can be used to reference child nodes multiple levels down the tree.

The selection node can exist anywhere in a tree path: At the beginning, in the middle, or as the last node. If multiple selection nodes are present in the tree path, a condition can be attached to each selection node. The query engine performs filtering starting at the top of the tree. If a selection node returns multiple nodes based on the condition, each node is processed further at the next tree path level. This process is done recursively until all

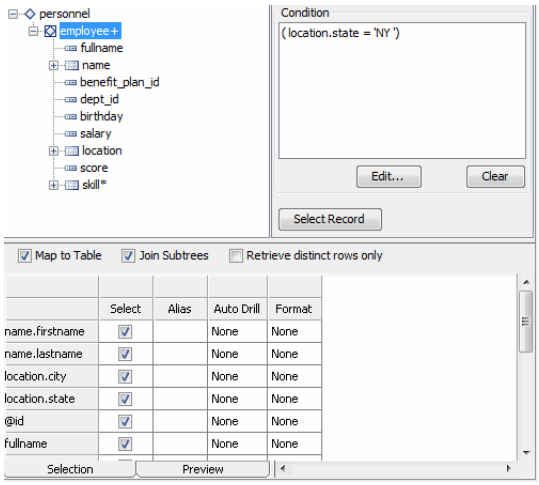
branches are processed. The collection of all selected sub-trees becomes the child nodes of a new root node.

Condition Walkthrough

Walkthrough

Next, create an example query to retrieve only employees from New York:

1. Create a query following the same steps as before to select the employee node and create the table construction rules.
2. Select the ‘Employee’ node so that the condition field is enabled.
3. Click on the ‘Edit’ button for the condition editor.
4. Expand the ‘location’ node, select the ‘state’ field, and click ‘Add’. Then fill in the condition to read “is equal to NY”.
5. Click ‘OK’ to accept the condition.





6. Click the ‘Save’ button on the main tool bar to save the condition.

The condition could also be entered manually in the condition text area as “location.state = ‘NY’”.

If you run the query, the result will only contain information on employees from NY.

Figure 4. Filtered Query Output

name.lastname	name.firstname	location.city	location.stat	birthday	salary	dept_id	score
Jacques	Smita	Middletown	NJ	1972-4-15	68000	1	3
Shyko	Sherri	New York	NY	1968-9-15	58000	2	5
Hartel	Sanford	Boston	MA	1975-10-15	38000	3	3
Gallaher	Samuel	Boston	MA	1968-10-15	57000	3	4
Lotzer	Robert	New York	NY	1969-8-15	65000	2	1
Hartnack	Rebecca	East Brunswick	NJ	1963-1-5	62000	1	4
McDonald	John	New York	NY	1967-10-15	88000	5	2

4.10.2 Querying a Text Data Source

Walkthrough

A text query specifies how the text file defined by the data source should be converted into a table. To create a query for the 'OldCRM' test data source, follow the steps below:

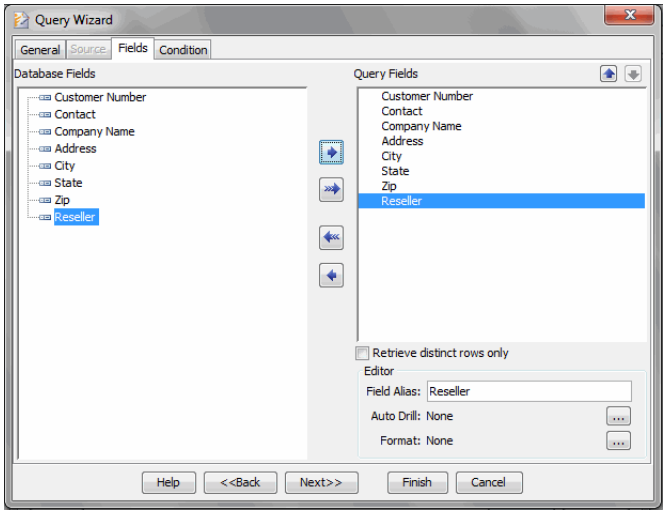
1. Click the 'New' button in the Style Studio toolbar. This opens the 'New Asset' dialog box.

2. In the left panel, select the 'Query' node. In the right 'Types' panel, select the 'OldCRM' data source, and click 'OK'. This opens the 'Query Wizard' dialog box.

3. Select the **General** tab. In the 'Name' field, enter the following query name: "Customer_List".

4. Select the **Source** tab (if enabled). Choose one of the requests that you have defined in the data source definition. (See *Text Data Source* for more information.)

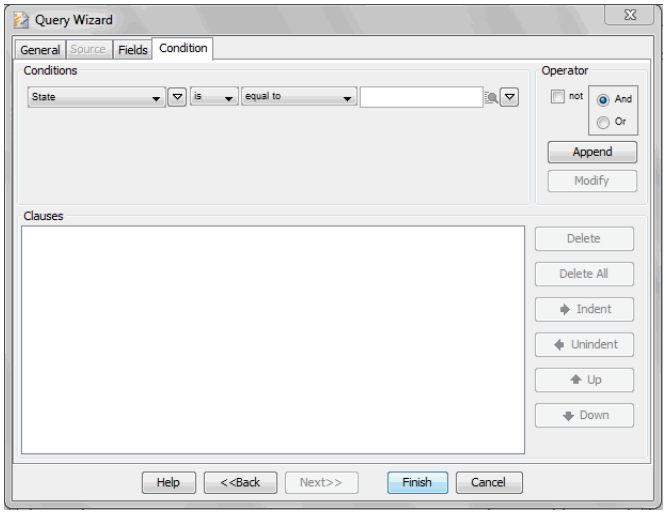
5. Select the **Fields** tab. Choose the database fields you want to include in the query.



6. (Optional) Select the **Condition** tab. Click a node in the tree to filter, and press the 'Add' button. For example, to select records for a particular state, select the `State` field.

7. Use the menus to specify the first part of a condition. For example:

```
[State][is][equal to]
```

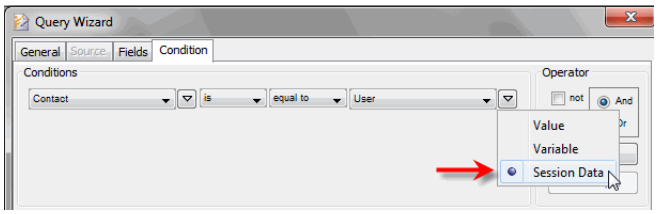


8. To use a value in the right-hand side of the condition, click the “arrow” button at the right side of the condition, and select the ‘Value’ option. Then enter the desired value in the text field.

9. To use a variable in the right-hand side of the condition, click the “arrow” button at the right side of the condition, and select the ‘Variable’ option. In the adjacent ‘Name’ field, enter a name for the parameter.

   This is the name (case sensitive) by which you will refer to the parameter in hyperlinks, scheduled tasks, scripts, etc. If the condition operator is “one of”, then the variable represents an array. Otherwise, it is a scalar value.

10. To use session data in the right-hand side of the condition, click the “arrow” button at the right side of the condition, and select the ‘Session Data’ option.



Note that the adjacent menu automatically chooses the ‘equal to’ or ‘one of’ operator to match your selection.

   In the menu, choose one of ‘User’, ‘Roles’, or ‘Groups’. These parameters return information about the user who is currently accessing the query; respectively, the user name, the array of roles to which the user belongs, and the array of groups to which the user belongs.

11. Press ‘Finish’ to exit the Wizard.



12. Click the ‘Save’ button in the Style Studio toolbar to save the query.

Figure 5. Text Query Output

Customer Number	Contact	Company Name	Address	City	State	Zip	Reseller
2	Lisa Smith	Daycare One	123 Summer Rd.	Pitterson	NJ	09882	false
3	Ana Lincoln	Eastern Data	23 Tasaga Ave	Aachen	TX	03887	true
4	Patrico Brown	Ernst Handel	88 Rio Grand Ave	Boulder	CO	38923	true
5	Steven Devon	George Services	389 Jardim Rd	Baltimore	MD	82938	false
6	Aria Duz	Familia Computer Services	5 Baker's Lane	Chicago	IL	39283	true
7	Lino Rodrigze	Old World Insurance	1230 Hoes Lane	New York	NY	08854	false
8	Richard	Quick Stop	71 Mulbery St	Denver	CO	98839	false

See Also

Text Data Source, for information on how to configure a Text data source.

4.10.3 Querying a Java Object Data Source

Once a data source is defined, the data source can be used to create Object queries. The query-building process is identical to the process for XML queries; the output of the Java Object is mapped to hierarchical meta-data, which can be selected and filtered using the same mechanism as for other hierarchical data sources.

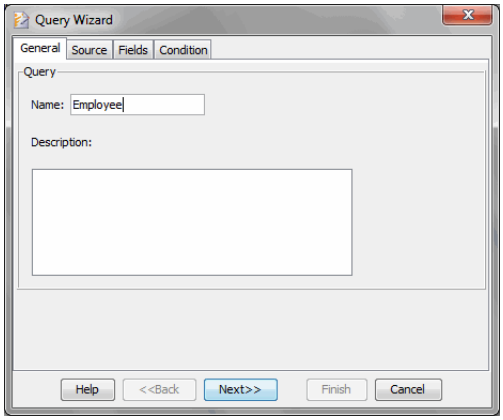
A query based on the Object data source is executed in the following sequence:

1. A Data Loader object is instantiated if none have been created yet.
2. The request method is invoked with the parameter values either from the data source definition or as were input by the user.
3. The collection of object return values is parsed into an object tree based on introspection.
4. Any user defined filtering and selection in the query is applied to produce the final result set.

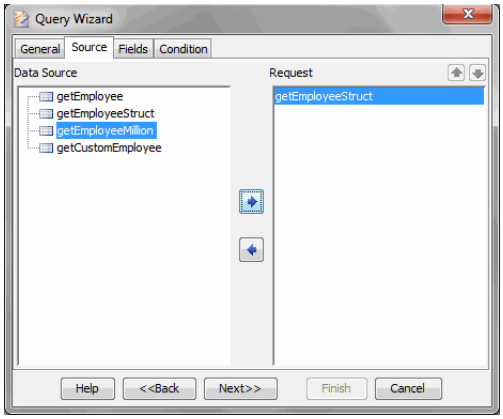
To define a query on the Object data source that you created in *Defining a Java Object Data Source*, follow the steps below:



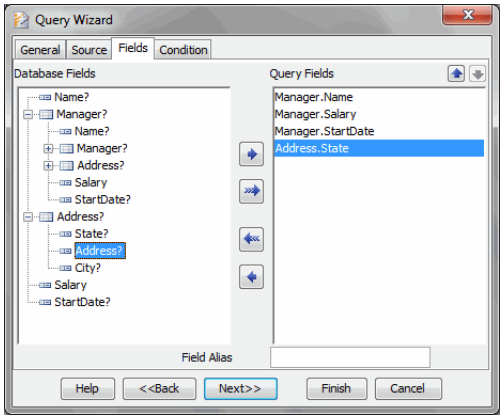
1. Click on the ‘New Query’ button to create a new query.
2. Type in “Employee” as the name of the query. Select ‘Object’ as the data source. Click ‘OK’.



3. Add the ‘getEmployee’ request and click ‘Next’.



4. Add ‘Manager.Name’, ‘Manager.Salary’, ‘Manager.StartDate’, ‘Address.City’, and ‘Address.State’ to the Report Fields list. Click ‘Finish’.





5. Click the ‘Save’ button to save the query.

See Also

Java Object Data Source, for information on how to configure a Java Object data source.

4.10.4 Advanced Query Concepts

This section discusses several advanced concepts.

Sub-Query

Sub-queries are queries used inside a query condition expression. The result of the sub-query is used when evaluating the expression. This functionality is only supported for hierarchical data sources like XML, SOAP, etc. You cannot use this feature with a non-hierarchical data source like JDBC. This concept also exists in SQL, but there are a few important differences:

- A sub-query can be used in an expression where a scalar or list value is expected. This is different from SQL sub-queries, which can only be used in a few specific types of expressions.
- A sub-query is referenced by the query name. The definition of the sub-query is not included in the expression. In SQL, the sub-query's definition is embedded in the expression where the sub-query is used. By referencing a sub-query by its name, the sub-query definition can be shared by more than one query, and the resulting query is easier to maintain.
- A sub-query can use a different data source from the parent query. This is very powerful in a hybrid data source environment. For example, an XML query can use a sub-query that retrieves its data from an EJB. This allows the application to ignore the underlying implementation of the data model and unify the data access interface.

Invoking a Sub-Query

A sub-query can be invoked from an expression using the 'query()' function.

```
name in (query('all names'))
```

In this example, the results returned by the query, 'all names', are used as the list value. The query must be defined in the query registry. If the sub-query expects parameters (user variables), the variable values can be passed to the sub-query in the function call.

```
name in (query('all names', state = state)
```

One or more parameters can be added to the call as needed. Each variable is passed in as a name-value pair. In this example, the name of the variable is 'state', and the value of the variable is the 'state' node value in the

current record. Any name values can be used as the variable value, including variables, sub-queries, nodes or attributes, and constant values.

Walkthrough: Designing the Sub-Query

The following example illustrates how to use a sub-query to get a list of values from an XML data stream and then using the query result in the condition of another query to retrieve a list of employees. The query retrieves a list of all employees with evaluation scores above a score of 3.

Walkthrough

First, create a query to retrieve the employee ID's of employees with scores above a certain value.

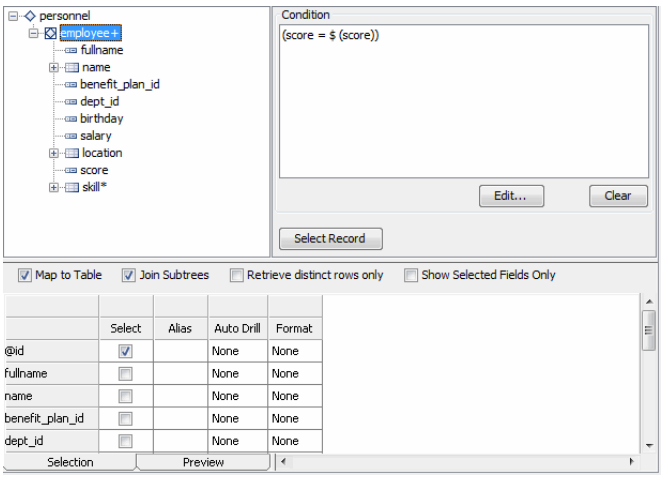
1. Create a query, "Scores", using the 'Personnel' data source.
2. Select the 'default' request type.
3. On the **Fields** tab, select the 'id' field, and click the 'add' button.
4. On the **Condition** tab, select the 'score' field, click the 'add' button, and set the condition to:

   ```
   score >= $(score)
   ```

5. Click 'Finish'.

   Notice that the 'Map to Table' option is selected. It would not be necessary to map the result to a table if the return value was a child node. In this case, however, you need to return the employee ID, which is an attribute in the employee record. Attributes can only be returned as a table column.

This query returns a table with a single column, the employee ID. The score comparison is done against the 'Score' variable.



Walkthrough: Designing the Main Query

Next, create a query to retrieve the detailed employee information using the sub-query as the selection criteria.



1. Create a query, “Above avg employees”, using the ‘Personnel’ data source.
2. Select the ‘default’ request type.
3. Using the **Fields** tab, select: ‘name.lastname’, ‘name.firstname’, ‘location.city’, ‘location.state’, and ‘score’.
4. Click ‘Finish’.
5. Select the ‘employee’ node and enter the following condition in the ‘Condition’ text area.

```
@id in(query('Scores',score=3))
```

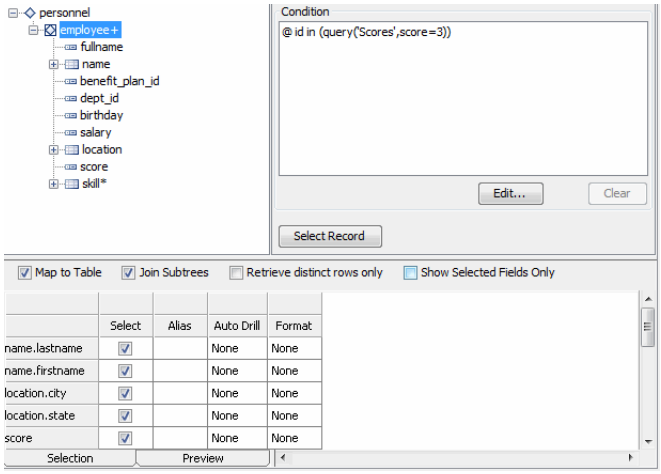


Figure 6. Query Output

name.lastname	name.firstname	location.city	location.state	score
Johnson	Ellen	East Brunswick	NJ	5
Hartnack	Rebecca	East Brunswick	NJ	4
Jacques	Smita	Middletown	NJ	3
Oakley	Frank	Middletown	NJ	5
Shyko	Sherri	New York	NY	5
Jacob	Jeffrey	New York	NY	5
Hartel	Sanford	Boston	MA	3

List Comparison for Queries using Hierarchical Meta Data

The syntax and grammar of the query condition expression is covered in Appendix A: *Hierarchical Query Condition Expression Syntax*. The following is an example.

If you want to get a list of all employees with C++ programming skill, you can add a condition to the employee node to select only people with C++ on their skill list. However, since each employee may have multiple skills linked in the employee record, a simple string comparison does not give the correct result.

```
skill = 'C++ programming'
```

This condition would work if all employees only had one skill listed. If the C++ programming were listed as the second skill of an employee, the comparison would return false because a list of skills is converted to a scalar value by using the value of the first child.

You need an expression to compare the skill list with a value, and the comparison should be true if any value on the list matches the skill. The following list comparison expression returns the correct result.

```
'C++ programming' = any skill
```

Walkthrough

Next, use this feature to create a query to retrieve all C++ programmers from the employee database.



1. Create an XML query as before, with the name “C++ Programmers”.

2. Select the ‘Employee’ node and click on ‘Select Record’ to make it the selection tree path.

3. Enter the condition expression in the condition text area:

```
'C++ programming' = ANY skill
```



4. Select ‘Save’ (on the main toolbar) to save the expression.

5. Select the ‘Map to Table’ option, then click ‘Preview’.

name.lastname	name.firstname	location.city	location.state
Johnson	Ellen	East Brunswick	NJ

List Pattern Matching for Queries using Hierarchical Meta Data

The syntax and grammar of the query condition expression in covered in Appendix A: *Hierarchical Query Condition Expression Syntax*.

This section introduces a slightly more difficult problem. Suppose you want to get a list of all employees with any programming skill, regardless of the particular programming language. Since there are no fields in the XML structure that specify this information, it will have to be deduced from the skill description.

Assuming all programming skills end with ‘Programming’, you can find employees with programming skills by matching the skills against a regular expression pattern, ‘.programming’. The following is a list pattern matching expression:

```
any skill match '.*programming'
```

Walkthrough

Follow the following steps to create an ‘All Programmers’ query.



1. Create an XML query as before, with the name “All Programmers”.

2. Select the ‘Employee’ node and click on ‘Select Record’ to make it the selection tree path.

3. Enter the condition expression in the condition text area:

```
ANY skill match '.*programming'
```



4. Select ‘Save’ (on the main toolbar) to save the expression.

5. Select the ‘Map to Table’ option, then follow the same procedures to map the result sub-tree to a table.

6. Click ‘Preview’.

skill	name.lastname	name.firstname	location.city	location.state
Java programming	Johnson	Ellen	East Brunswick	NJ
C++ programming	Johnson	Ellen	East Brunswick	NJ
VB programming	Hartnack	Rebecca	East Brunswick	NJ
C programming	Jacques	Smita	Middletown	NJ
Java programming	Oakley	Frank	Middletown	NJ
Accounting	Shyko	Sherri	New York	NY
Human Resource	Shyko	Sherri	New York	NY

APPENDIX A: Hierarchical Query Condition Expression Syntax

The query condition expression grammar for hierarchical data sources (XML, Web Service, Java Object) is based on SQL condition syntax, and most of the expressions have a construction identical to their SQL counterparts. Like SQL, all reserved words are case-insensitive. However, all names, including variables and child nodes, are case-sensitive. The following sections cover the complete list of conditional expressions and give a few examples of advanced usage.

A.1 Named Values

In SQL's conditions, the column values can be used in expressions for calculation or comparison. Similarly, Style Studio condition expressions can reference values on the data tree. However, since the hierarchical data model used by Style Studio supports much richer organization of data, the expressions support a few more types of data references.

A.1.1 Child Node Values

The selection node and all of its child nodes can be referenced by the conditional expression. The value of the selection node can be referenced using the reserved word, 'this'.

```
this is null
```

Its immediate child nodes can be referenced by their names,

```
salary > 50000
```

The descendents of the child nodes can be referenced with a node path. A node path is constructed by appending the node names from the child node, and all of the nodes leading to the referenced node, with the node names separated by dots. For example, the child of 'location' node can be referenced using:

```
location.city = 'New York'
```

Using the same algorithm, a path can be constructed for arbitrary levels down the sub-tree,

```
child1.grandchild2.grandgrandchild3 like 'N%'
```

If the referenced node can have multiple instances, the node value is extracted from the first instance of the node. For example, the 'Skill' node in the 'Employee' tree can have more than one instance. An employee can have multiple skills. A 'Skill' node describes each. If the skill child is used in an expression, it refers to the first occurrence of the 'skill' node.

```
skill like 'Java%'
```

It is possible to specify the exact instance by adding an index parameter to the node name.

```
skill like 'Java%'
```

The index is zero based. If the specified instance does not exist, the node reference returns a null value.

Node names used directly in an expression must not contain any spaces or other special characters. Like identifiers in a programming language, a node name must start with an alphabetic character or '_', followed by zero or more alphanumeric characters. In case a node name does not conform to this requirement, or the name collides with a reserved word, the node can be referenced using the 'node' function,

```
node('first name') = 'John'
```

This is functionally equivalent to the direct reference to a node, but allows any node name to be used regardless of the characters in the name.

A.1.2 Node Attributes

Each node may have zero or more attributes. The attributes are not child nodes of the node. Each attribute contains a scalar string value. The attribute of a node can be referenced using the attribute name appended to an '@' sign.

```
@id = '339-928-9877'
```

When the attribute name is used without any qualification, as in the above example, it refers to the attribute in the current node. Appending the attribute name to the node path can reference the attribute values of any child node:

```
child1.grandchild.@type = 'integer'
```

The 'node' function can also be used to reference an attribute:

```
node('child1.second child.@name') = 'John'
```

A.1.3 Variables

A variable can be added to an expression where a value is expected. The variable does not need to be declared. Whenever a variable notation is encountered, the variable is implicitly declared. If a variable is used without any further definition, it defaults to the string type, and the end user enters its value when the query is executed.

```
location.state = $(state)
```

The name of the variable can be quoted in the parentheses. This allows reserved words to be used as the variable name.

```
price = $('max')
```

If the quote were not used, the expression parser would generate an error because the word ‘max’ is used for the aggregate function and is not recognized as a regular variable name.

All variable references to the same name in all expressions used in a query are considered to be the same variable. Variables are never shared across queries, regardless of their names.

A variable can be formally defined in a query. The variable definition sets the follow variable attributes:

- **Alias**
  Variable label used when prompting end users for the variable value.
- **Type**
  Variable data type. The type determines the component used for entering the variable value and the type of the entered value. For example, if a variable is defined as a date type, a date combo box is used to enter the variable value and the entered value is a `java.util.Date` object.
- **Value**
  Variable default value.

A variable can also be defined as a query-based variable. In this case, the value of the variable is derived from the result of another query and the end user is not prompted for the value.

A.2 Constant Values

There are four basic types of constants: string, number, Boolean, and date.

String

A string value is a quoted character sequence. Both single quotes and double quotes are accepted.

```
city = 'New York'
  or
city = "New York"
```

If a single or double quote is part of the string value, it needs to be escaped if the value is quoted using the same character. To escape a quote character, repeat the character twice, e.g.,

```
'Seven O''Clock'
```

Number

A number can be either an integer or a double.

```
age > 55
```

A double number can be entered using either decimal notation, or scientific notation,

```
ratio < 2.5
```

Numeric comparison is always done in double values. Therefore, an integer 50 does not equal a double 50.1. All arithmetic computation is also done in doubles. All values are converted to a double before the computation is performed. A division of two integer numbers results in a double value. No rounding is done regardless of the type of the operands.

Boolean

A Boolean constant is specified using 'true' or 'false'. The case is insignificant.

```
instock = true
```

Date

There is no date constant in an expression. Instead, converting a text representation of a date to a date object can create a date value. The conversion is done using the `to_date()` function.

```
birthday > to_date("1960-12-31")
```

The default date format is the same as for SQL, `yyyy-MM-dd`. An alternative date format can be supplied in the second argument of the function call:

```
birthday > to_date("12/31/60 12:00:00","MM/dd/yy HH:mm:ss")
```

The format should conform to the format defined in the java.text.SimpleDateFormat class:

SYMBOL	MEANING	PRESENTATION	EXAMPLE
G	era designator	Text	AD
y	year	Number	1996
M	month in year	Text & Number	July & 07
d	day in month	Number	10
h	hour in am/pm (1~12)	Number	12
H	hour in day (0~23)	Number	0
m	minute in hour	Number	30
s	seond in minute	Number	55
S	millisecond	Number	978

E	day in week	Text	Tuesday
D	day in year	Number	189
F	day of week in month	Number	2 (2nd Web in July)
w	week in year	Number	27
W	week in month	Number	2
a	am/pm marker	Text	PM
k	hour in day (1~24)	Number	24
K	hour in am/pm (0~11)	Number	0
z	time zone	Text	Pacific Standard Time
‘	escape for text	Delimiter	
“	single quote	Literal	‘

A.3 Simple Expressions

This section lists various supported expressions.

Arithmetic Expressions

Numeric computations are always done using double values. The following operators are supported:

+	Addition
-	Subtraction
*	Multiplication
/	Division
- (Unary)	Negation

Comparison Expressions

The expression ‘is null’ can be used to compare a value to null.

Comparisons can be done between any two values. If the two values have the same type, the comparison is done according to the type. If two values are of different types, then both values are converted to strings and compared using text comparison.

<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
= or ==	Equal to
<> or !=	Not equal

Logical Expressions

The conditional expressions are all short-circuit logic operations. In the ‘and’ expression, the right-hand operand is only evaluated if the left-hand operand is true. In the ‘or’ expression, the right-hand operand is only

evaluated if the left-hand operand is false.

And	And operation
Or	Or operation
Not	Logic negation

The operands of logic expressions can be any type. If an operand is not a Boolean value, it is converted to a Boolean. If the value is null, it is converted to a false value. Otherwise, it is converted to a true value.

A.4 SQL Predicates

The query condition also supports the other predicate expressions in defined SQL.

A.4.1 Between Operator

The ‘between’ comparison is shorthand for a ‘greater than or equal to’ and a ‘less than or equal to’ expression.

```
salary between 30000 and 40000
```

This is equivalent to the following condition.

```
salary >= 30000 and salary <= 40000
```

A negative ‘between’ comparison can be created using ‘not between’.

```
salary not between 30000 and 40000
```

A.4.2 In Operator

The ‘in’ operator tests if a value is one of the values in a given list. In the simplest form, the value list is given directly in the condition. Each value in the list can be either a constant or an expression that returns a scalar value.

```
state in ('NY', 'NJ')
```

In SQL, the list can be the result of a sub-query. This is also supported in Style Studio queries. However, the sub-query has a different syntax. The details for using sub-queries are discussed later.

```
state in (query('tristates'))
```

In addition to supporting sub-queries in the ‘in’ operation, the list value can come from any named value, such as a variable or a sub-selection.

Like the ‘between’ operator, there is a shorthand for adding a logic negation to the condition.

```
state not in ($(tristates_variable))
```

A.4.3 SQL Pattern Matching Operator

SQL has a very simple pattern-matching operator. The ‘like’ operator supports two pattern characters:

%	Match any character sequence
?	Match any single character.

```
company like 'I%'
```

This condition matches any string starting with an ‘I’ character.

```
skill like 'C??'
```

This condition matches any string starting with a ‘C’ character, followed by two arbitrary characters. Similar to the ‘in’ operator, a logic negation can be added with a short hand:

```
skill not like 'C%'
```

A.4.4 Existence Operator

In SQL, the existence test is applied to a sub-query. The condition tests if any result is returned by the sub-query. In a Style Studio query condition, a sub-query is treated the same as any named value. Therefore the existence test can be applied to any named value.

The result of the condition depends on the type of the data in the named value. If the value is a table, the existence test checks for the existence of any rows in the table. If the table is empty, the test returns false. Otherwise, it returns true.

If the value is a sequence (a node holding the collection of child nodes with the same type), the test checks for the existence of any child nodes in the sequence. If the sequence is empty, the test returns false. Otherwise, it returns true.

For all other values, the test is true if the value is not null.

```
exists query('salary over 200000')
```

The existence test can also be applied to a node. For example, to check if the “link” node exists:

```
exists link
```

A.4.5 List Comparison

A value can be compared to a list of values. The result of the comparison depends on what type of comparison is declared. Two list comparison types are supported:

- **Any**
  The comparison is successful if the value compares to true with any one of the values on the list.
- **All**
  The comparison is successful if and only if the value compares to true with all values on the list.

The following comparison tests if the price is greater or equal to all prices in the group:

```
price >= all query('all prices')
```

The ‘query’ function executes a sub-query and returns the results. It is covered in more detail in a later chapter.

The next condition tests if the state is in one of the states in the list:

```
state = any query('tristates')
```

A.5 Regular Expressions

Regular expressions are among the most powerful pattern matching mechanism, used extensively in text processing (often by Perl and AWK scripts).

```
Regular expression operations are relatively expensive compared to
standard string comparisons. Use them only if required.
```

Regular expressions are supported in expressions for matching a string value against a pattern. The pattern matching can be done on one string value or on a list of values.

A.5.1 Regular Expression Operator

The Style Studio query condition adds support for regular expression pattern matching. Regular expression is a much more powerful pattern language than the simple patterns supported by the ‘like’ operator.

The regular expression supported by Style Studio conforms to Perl5 regular expression syntax. For more details on the regular expression, please consult the Perl5 documentation.

Language Summary

The following is a summarization of the regular expression syntax.

```
Alternatives separated by |
Quantified atoms
          {n,m} Match at least n but not more than m times.
   {n,} Match at least n times.
   {n} Match exactly n times.
   * Match 0 or more times.
```

```
   + Match 1 or more times.
   ? Match 0 or 1 times.
a . matches everything except \n
a ^ is a null token matching the beginning of a string or
line (i.e., the position right after a newline or right
before the beginning of a string)
a $ is a null token matching the end of a string or line
(i.e., the position right before a newline or right after
the end of a string)
Character classes (e.g., [abcd]) and ranges (e.g., [a-z])
Special backslashed characters work within a character class
(except for backreferences and boundaries).
\b is backspace inside a character class
Special backslashed characters
\b null token matching a word boundary (\w on one side and
\W on the other)
\B null token matching a boundary that isn't a word boundary

\A Match only at beginning of string
\Z Match only at end of string (or before newline at the
end)
\n newline
\r carriage return
\t tab
\f formfeed
\d digit [0-9]
\D non-digit [^0-9]
\w word character [0-9a-z_A-Z]
\W a non-word character [^0-9a-z_A-Z]
\s a whitespace character [ \t\n\r\f]
\S a non-whitespace character [^ \t\n\r\f]
\xnn hexadecimal representation of character
\cD matches the corresponding control character
\nn or \nnn octal representation of character unless a
backreference.
a \1, \2, \3, etc. match whatever the first, second, third,
etc. parenthesized group matched. This is called a
backreference. If there is no corresponding group, the
number is interpreted as an octal representation of a
character.
\0 matches null character
Any other backslashed character matches itself
Expressions within parentheses are matched as subpattern
groups and saved for use by certain methods.
```

A.5.2 List Regular Expression Matching

The ‘match’ operator matches a string value with a regular expression.

```
company match 'I.*'
```

Similar to the ‘like’ operator, a logic negation can be added with a ‘shorthand’ method.

```
company not match 'I.*'
```

Like the list comparison expression, a list of values can be matched against a regular expression. The comparison can be defined as either ‘Any’ or ‘All’.

```
any skill match 'C.*Programming'
```

For the ‘Any’ type list matching, the result is true if any value in the list matches the regular expression. For an ‘All’ type list matching, the result is only true if all values in the list match the regular expression.

A.6 Aggregate Functions

Aggregate functions calculate values based on a list of values. The function can be applied to any value that contains a list, including sub-queries and sub-selections. In all functions, if the list value is null, it is treated as an empty list.

Sum

The *Sum* function calculates the total of the values in a list.

```
revenue > sum(query('ne-sales'))
```

Like all computational expressions, the values in the list are all converted to doubles to calculate the total. If any value on the list is not a number or a string convertible to a number, the value is converted to a zero.

Avg

The *Avg* function calculates the average of the values in a list.

```
price > avg(filter('product.price'))
```

The average is calculated as a double value. All items on the list, regardless of whether they are valid numbers, are counted into the average. All non-numeric values are treated as zero.

The *Min* function returns the minimum value on a list.

```
price = min(filter('product.price'))
```

Max

The *Max* function returns the maximum value on a list.

```
price = max(filter('product.price'))
```

Count

The *Count* function returns the number of items on a list.

```
count(query('PDAs')) > 10
```

A.7 Sub-Query

SQL conditions allow a sub-query to be used in certain expressions. For example, a sub-query can be used in the ‘in’ expression to serve as the list value. This concept is supported in the Style Studio conditions.

The SQL sub-query is specified as a select statement inside a SQL condition. Using Style Studio, you can specify conditions on query variables such that they are dependent on the results from the sub-query. For example:

```
select * from orders where orders.state in (select
customers.state from customers)
```

The result of the sub-query is interpreted according to the context.

If a scalar value is expected in the expression, the sub-query result (a sub-tree) is converted to a value using the following rules:

- If the root node is a sequence node, the value of the first child is returned.
- If the root node value is not null, the value is used as the scalar value.
- If the root node value is null and has child nodes, the value of its first child is returned.
- Otherwise, the value is null.

If a list is expected in the expression, the sub-query result is converted to a list using the following rules:

- If the root node is a table node, the first column of the table is converted to a list.
- If the root node is a sequence node, the sequence is converted to a list by converting each child of the root to its scalar value.
- Otherwise, a single value list is created using the scalar value of the root.

A sub-query call can be used in any place in a condition expression where a value is expected.

A.8 Sub-Selection

Previously you have learned how to define a selection tree path using the query panel. The tree path is saved as part of a query, and is used by the query to select sub-trees from the data stream. The same tree path selection mechanism can be used inside a query condition.

A tree path can be used to select nodes from the current sub-tree.

```
name in (filter('employee[name = "John
Smith"].subordinates.name'))
```

The filter() function can be used where a scalar or list value is expected, such as a sub-query or a variable. The result of the filtering is converted to a scalar or list value using the same algorithm as was used for the result of a sub-query.

A tree path is constructed by concatenating the node names from the root of the current sub-tree, separated by a dot. For example, if a condition is attached to the ‘employee’ node, a tree path to select all skill nodes is:

```
$(skill) in filter('employee.skill')
```

Sub-selection is different from referencing child nodes by name or node path. If ‘employee.skill’ is used in an expression, it refers to the skill list of the current employee. Since the condition is evaluated for each employee individually, the list only contains the skills of one employee. On the other hand, a filter always works on the entire list. The ‘filter(‘employee.skill’)’ expression returns a list of all of the skills contained on the employee list.

The other difference between a selection tree path and a regular node name path is that conditions can be added to the tree path. The condition is used to filter the nodes to be included in the sub-tree. For example, if you want to retrieve all programming skills, you can match the skill with a regular expression:

```
$(skill) in filter('employee.skill[this match
".*Programming"]')
```

As with SQL sub-queries, using a sub-tree selection in a condition expression is quite expensive. This option should only be used when no other construct achieves the same purpose.

APPENDIX B: OLAP Server Setup

This section provides various notes on OLAP server configuration.

See Also
Non-Relational Data Sources, for information on using OLAP data sources.

B.1 Microsoft SQL Server 2000

1. Microsoft Analysis Services should be installed.

2. Install Microsoft XMLA for Analysis, version 1.1 or later.

3. The file *datasource.xml* in the *<XMLA installation directory>/config* directory is configured to use `localhost` by default. If the OLAP server is on another machine, the file must be reconfigured. Multiple data sources can be included in this file.

4. The file *msxisapi.dll* in the *<XMLA installation directory>/isapi* directory must be made available in a web server. Set up a virtual directory (e.g., *xmla*) on the web server which points to the *isapi* directory. When the URL (e.g., `http://localhost/xmla/msxisapi.dll`) is entered in a browser, it should return a SOAP message. If it fails, make sure the end user has been granted permission to execute scripts.

B.2 Microsoft SQL Server 2005

1. Microsoft Analysis Services 2005 should be installed.

2. The *msmdpump.ini* file in the *{SQLServer}\MSSQL.1\OLAP\bin\isapi* directory is configured to use `localhost` by default. If the OLAP server is on another machine, the file must be reconfigured. Multiple data sources can be included in this file.

3. The *msmdpump.dll* file in the '*{SQLServer}\MSSQL.1\OLAP\bin\isapi*' directory must be made available in a web server. Set up a virtual directory (e.g., *xmla*) on the web server which points to the *isapi* directory. When the URL (e.g., `http://localhost/xmla/msmdpump.dll`) is entered in a browser, it should return a SOAP message. If it fails, make sure the end user has been granted permission to execute scripts.

Advanced Data Mashup

The Data Worksheet is a powerful and easy-to-use tool that enables you to rapidly build complex queries for demanding tasks such as “what-if” and cross-domain analysis.

1 Introduction

The Data Worksheet is the second level of Data Block™ technology, sitting on top of the atomic data blocks (data models, queries) built in Style Studio. For more information on creating atomic data blocks, see *Advanced Data Modeling*.

The Data Worksheet is a powerful and easy-to-use tool that enables you to rapidly build complex datasets for tasks such as "what-if" and cross-domain analysis. Using the simple mouse-driven interface, you can perform ad hoc query and data mashup with diverse data sources by leveraging the atomic data blocks provided for enterprise sources and even bringing in your local data. You can manipulate these datasets into versatile Data Blocks. These Data Blocks can then be used in publishable reports, analytic dashboards, and as components of other data worksheets.

See Also
Assets: Reusable Worksheet Components, for information on creating Data Blocks and using assets in Worksheets.

1.1 Style Studio and Visual Composer

You can create and edit Worksheets using either the desktop *Style Studio* or the web-based *Visual Composer*. The Style Studio and Visual Composer tools provide identical functionality, although there are some minor differences in appearance. Both tools save assets to the same location, and both have access to the same saved assets.

```
Note: The screen images shown in this guide are generally from the
Visual Composer.
```

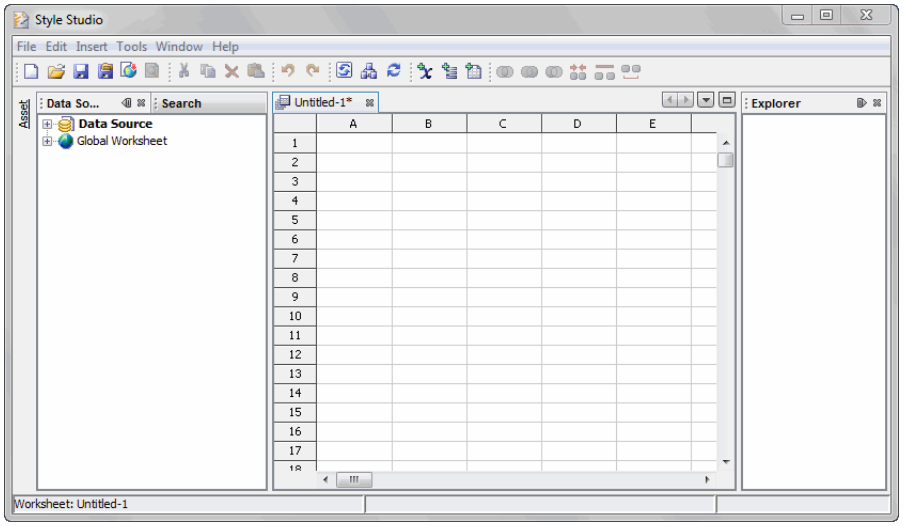
You can use either tool to create and edit Worksheets. Developers may find it more convenient to work with the desktop Style Studio, while Viewsheet users may find it more convenient to work with the web-based Visual Composer.

See Also
Saving an Asset, for information about asset storage.
Editing Data Worksheets, for information on opening and creating Worksheets.

1.1.1 Editing a Data Worksheet in Style Studio

To edit a Worksheet in Style Studio, open the **Asset** panel and expand the Worksheet node. Double-click the desired worksheet.

To create a new Worksheet in Style Studio, click the 'New' button in toolbar. This opens the 'New Asset' dialog box. Select the 'Worksheet' node in the left-side tree, and click 'OK'.



You can now proceed to edit the Worksheet. See *Editing Data Worksheets* for more information.

1.1.2 Editing a Data Worksheet in Visual Composer

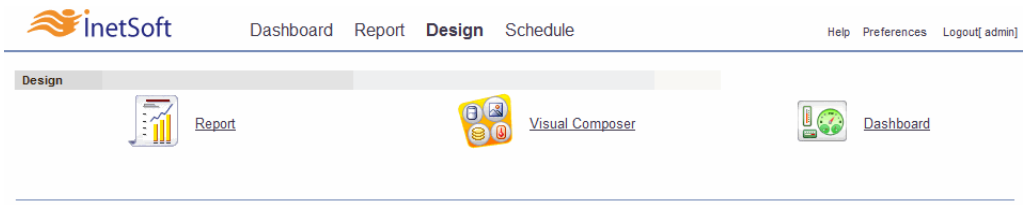
To launch the web-based Visual Composer, follow these steps:

1. Open a web browser, and log in to the User Portal.

   The default URL for the User Portal is `http://hostname:8080/sree/Reports`, where '`hostname`' is the computer running the Portal software. Consult your administrator for the correct address and login information.

2. Click the **Design** tab at the top of the User Portal.

3. Click the 'Visual Composer' link under the **Design** tab. This launches the Visual Composer in a separate browser window.



Once you have launched the Visual Composer, you can create new Worksheets or open existing Worksheets for editing. See *Editing Data Worksheets* for more information.

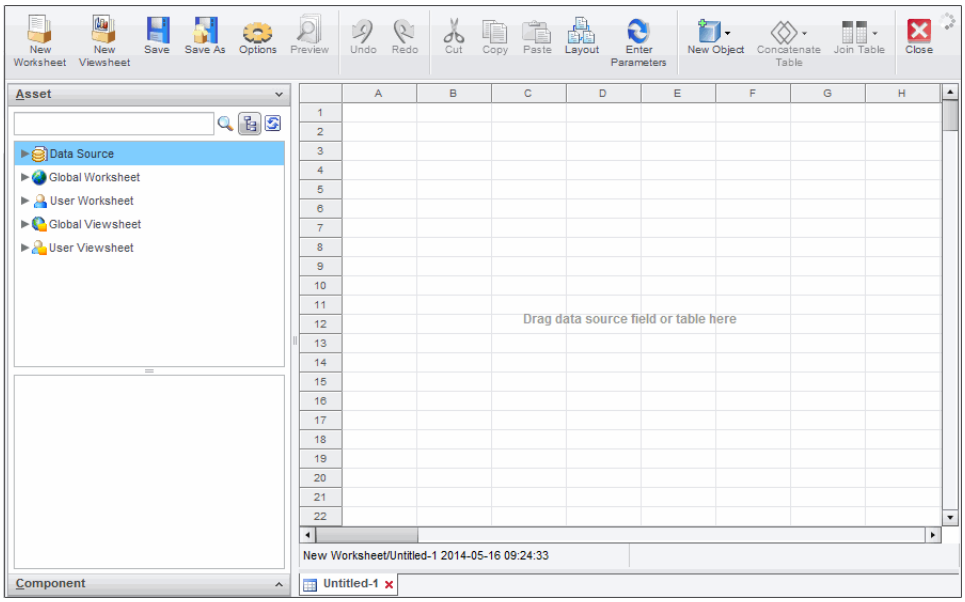
1.2 Editing Data Worksheets

This section provides basic information about working with Worksheets.

1.2.1 Creating a New Data Worksheet



To create a new Worksheet (in Style Studio or Visual Composer), click the 'New Worksheet' button in the toolbar. This opens a blank Worksheet named 'Untitled'.



See Also

Assets: Reusable Worksheet Components, for adding content to a Worksheet.

Appendix A:*Toolbar Buttons*, for information on the Worksheet toolbar.

1.2.2 Creating a Local Worksheet

A *local Worksheet* is a Worksheet that is embedded within a report, rather than being stored in the common *asset.dat* repository file. Embedding a Worksheet data source within a report enhances report portability. However, because a local Worksheet is available only to the particular report in which it is embedded, embedding is most useful in cases when the Worksheet is unique to the report and will not be reused for other reports.

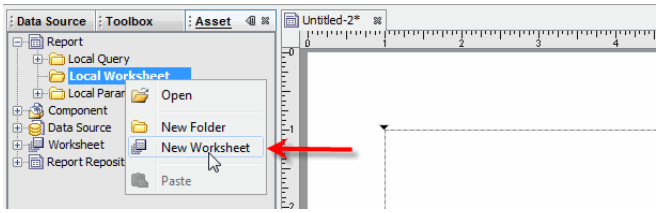
Creating a Local Worksheet

To create a local Worksheet, you must first open a report in Style Studio. Follow these steps:

1. In Style Studio, open the report into which you want to embed the local Worksheet.

2. Expand the Style Studio **Asset** panel.

3. Expand the 'Report' node, and right-click the 'Local Worksheet' folder. Select 'New Worksheet' from the context menu.



Note: The 'Local Worksheet' folder is only available when a report is open for editing in Style Studio.

4. Design the new Worksheet as desired, and save the Worksheet. (See the other sections of this guide for information on designing Worksheets.)

The saved Worksheet is displayed under the 'Local Worksheet' folder, and is embedded within the current report. You can bind this Worksheet to a report element in the same way that you bind any other data source. See *Walkthrough: Choosing the Binding Data Source* in *Advanced Report Design* for more information.

See Also

Creating a Local Query, in *Advanced Data Modeling*, for information on embedding a query.

Exporting a Local Worksheet to the Global Repository

To export one or more local Worksheets to the global *asset.dat* registry file so that they can be used in other reports, follow the steps below:

1. In Style Studio, open the report that contains the local Worksheet(s) that you want to export.

2. Expand the Style Studio **Asset** panel.

3. Expand the 'Local Worksheet' node in the tree.

4. Expand the desired target folder under the 'Worksheet' tree node.

5. To *copy* the Worksheet(s) to the desired folder under the 'Worksheet' tree node (and leave the originals as embedded local Worksheets), follow these steps:

   a. Select the desired Worksheet(s) in the 'Local Worksheet' folder, right-click, and select 'Copy' from the context menu.

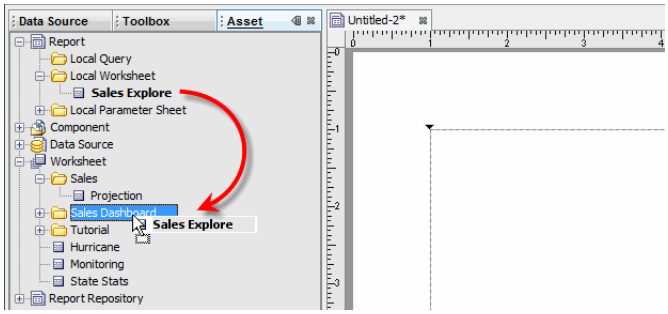
   b. Right-click the desired folder under the 'Worksheet' tree node, and select 'Paste' from the context menu.

Alternatively, drag-and-drop the Worksheet to the desired location while holding down the **Ctrl** key.

6. To *move* the Worksheet(s) to the desired folder under the 'Worksheet' tree node (and remove them from their embedding the report), follow these steps:

    a. Select the desired Worksheet(s) in the 'Local Worksheet' folder.

    b. Drag-and-drop the selected Worksheet(s) into the desired folder under the 'Worksheet' tree node.



The selected Worksheets are added to the common asset registry. Elements in the report that were previously bound to removed local Worksheets are automatically re-bound to the newly-exported global Worksheets.

Importing a Local Worksheet from the Global Repository

To import one or more Worksheets from the common *asset.dat* registry file into a specific report as local Worksheets, follow the steps below:

1. In Style Studio, open the report into which you want to embed the Worksheet(s).

2. Expand the Style Studio **Asset** panel.

3. Expand the 'Report' node, which contains the 'Local Worksheet' folder.

4. Expand the folder under the 'Worksheet' tree node that contains the Worksheet(s) that you wish to import.

5. To *copy* the Worksheet(s) to the 'Local Worksheet' folder (and leave the originals in the common asset registry), follow these steps:

    a. Select the desired Worksheet(s), right-click, and select 'Copy' from the context menu.

    b. Right-click on the 'Local Worksheet' folder, and select 'Paste' from the context menu.

Alternatively, drag-and-drop the Worksheet to the desired location while holding down the **Ctrl** key.

6. To *move* the Worksheet(s) to the ‘Local Worksheet’ folder (and remove them from the common asset registry), follow these steps:

   a. Select the desired Worksheet(s).

   b. Drag-and-drop the selected Worksheet(s) onto the ‘Local Worksheet’ folder.

The Worksheet is now displayed under the ‘Local Worksheet’ folder and embedded within the current report. You can bind this Worksheet to a report element in the same way that you bind any other data source. See *Walkthrough: Choosing the Binding Data Source* in *Advanced Report Design* for more information

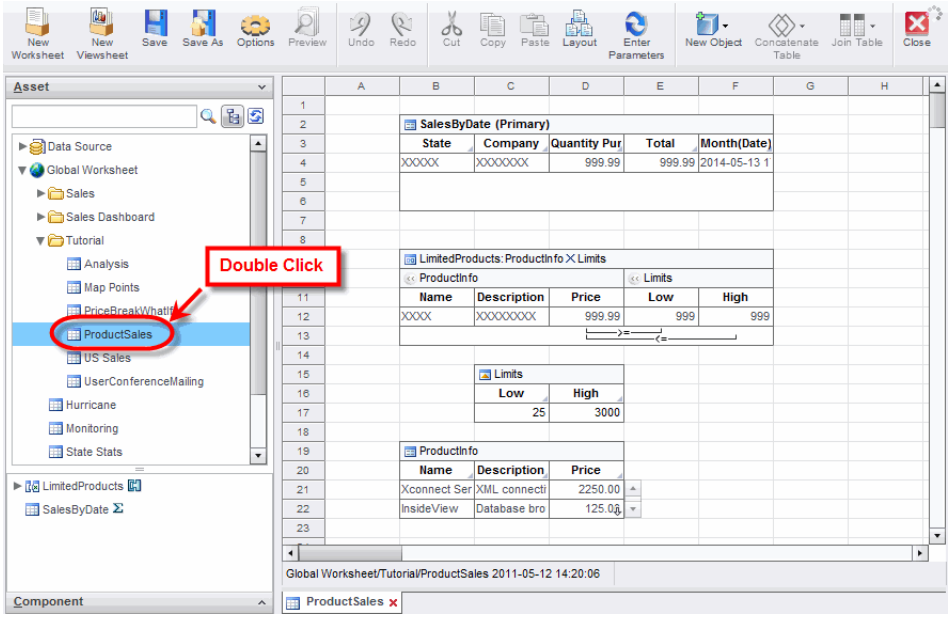
Report elements that were previously bound to the removed global Worksheet are automatically re-bound to the newly-created local version.

1.2.3 Opening a Data Worksheet

To open a saved Worksheet in Visual Composer (in Style Studio or Visual Composer), follow these steps:

1. Expand the ‘Global Worksheet’ or ‘User Worksheet’ node in the Asset Repository tree, and locate the desired Worksheet.

2. Double-click the Worksheet in the Asset tree, or right-click the Worksheet and select ‘Open Sheet’.



The Visual Composer loads the Worksheet into a new page, and displays the name of the Worksheet in the tab at the bottom. Visual Composer

allows you to keep many Worksheets (and Viewsheets) open simultaneously.

See Also

Assets: Reusable Worksheet Components, for information on editing a Worksheet.

1.2.4 Saving a Data Worksheet

This section discusses Worksheet storage and scope.

Saving a New Version of an Existing Worksheet



To save a new version of an existing Worksheet, click the ‘Save’ button in the toolbar, or press Ctrl-S on the keyboard. This replaces the older version with the current version, retaining the same name.



To save the new version of the Worksheet under a different name, click the ‘Save As’ button. This opens the ‘Save Worksheet’ dialog box. Then follow the instructions for saving a new Worksheet below.

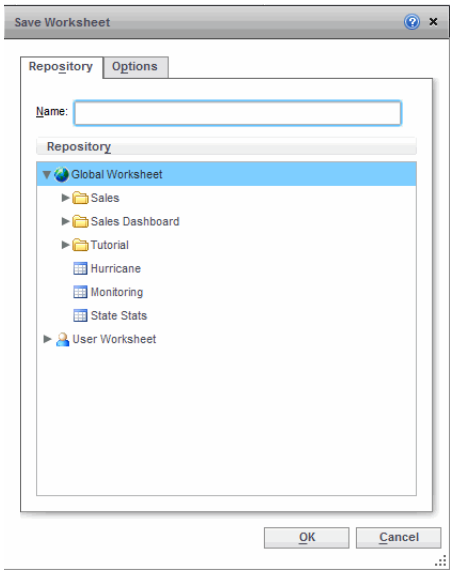
Saving a New Worksheet

To save a new Worksheet, follow these steps:

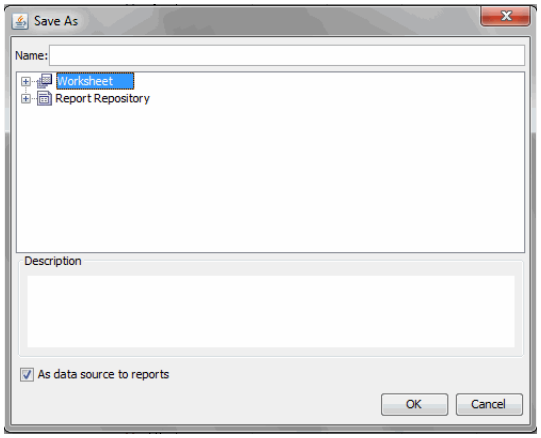


1. Click the ‘Save’ button in the toolbar, or press Ctrl-S on the keyboard. This opens the ‘Save Worksheet’ dialog box.

   The dialog box in Visual Composer appears as shown below:



   The dialog box in Style Studio appears as shown below:



2. Enter a name for the Worksheet in the ‘Name’ field. In Visual Composer, this is available under the **Repository** tab, e

3. Select a location in which to save the Worksheet:

   a. If you are using Visual Composer, select the appropriate scope for the saved Worksheet: ‘Global Worksheet’, ‘User Worksheet’, or a sub-folder. The scope in which you save the Worksheet determines the accessibility of the Worksheet to different users. See *Controlling Access to an Asset* for further details.

   b. If you are using Style Studio, select the appropriate scope for the saved Worksheet: ‘Worksheet’, a sub-folder under ‘Worksheet’, or a report under ‘Report Repository’. The scope in which you save the Worksheet determines the accessibility of the Worksheet to different users. See *Controlling Access to an Asset* for further details.

4. Select ‘As data source to reports’ to make the Worksheet result (i.e., the Primary asset) available to reports as well as Viewsheets. This will allow you to bind the Worksheet result to report elements such as tables, charts, and sections. In Visual Composer, this setting is available under the **Options** tab of the dialog box.

5. (Optional) If you are using Visual Composer, under the **Options** tab, set the ‘Design mode sample data size’ to the maximum number of rows that wish tables in the Worksheet to retrieve.

6. Click ‘OK’ to save the Worksheet.

The Worksheet is saved to the Asset Repository, and appears in the Asset Repository tree under the specified scope. Saved Worksheets are stored in a file called *asset.dat* in the SREE Home directory.

Example: Saving a global Worksheet

Create a new Data Worksheet that includes a Composition Table listing all of the orders from the states of NY and NJ, and save the Worksheet in the Asset Repository.

Follow these steps:



1. Create a new Worksheet.

2. In the Asset Repository, expand the 'Data Source' node, 'Orders' node, and 'DWS' node.

3. Drag the 'NJ Orders' query onto an empty cell on the Worksheet. This creates a new table, 'NJ Orders1'.

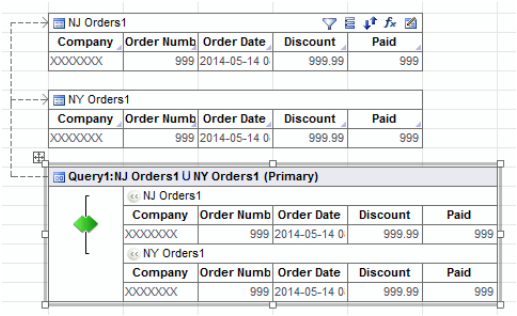
4. Drag the 'NY Orders' query onto an empty cell on the Worksheet. This creates a new table, 'NY Orders1'.

5. Select both tables by holding down the Ctrl key and clicking on the title rows of both tables.



6. Click on the 'Concatenate Table' button on the toolbar, and select 'Union'. This creates a new table, 'Query1', which is the union of the data from the 'NJ Orders1' and 'NY Orders1' tables.

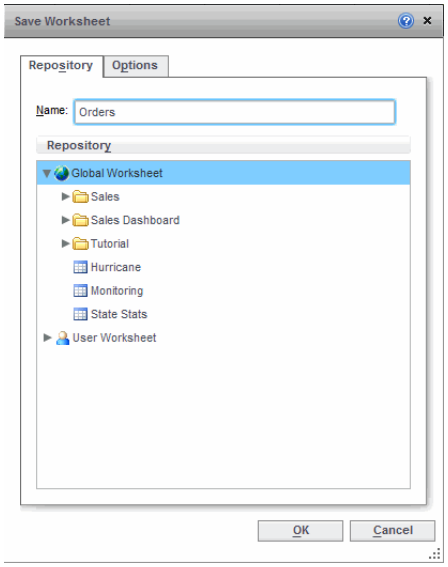
7. Set the 'Query1' table as the Primary Table by right-clicking on its title row and selecting 'Set as Primary'.





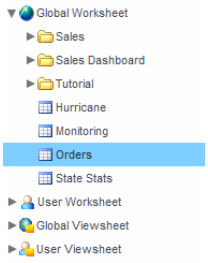
8. Click on the 'Save' button in the toolbar to save this Data Worksheet. This opens the 'Save Worksheet' dialog box.

9. Enter "Orders" for the Worksheet name, and select 'Global Worksheet' as the scope.



10. Click on the ‘OK’ button.

Note that the ‘Orders’ asset is saved under the ‘Global Worksheet’ node on the asset tree.



¢

See Also

Controlling Access to an Asset, for information about scope rules.

Using an Asset, for information about binding worksheet results to elements.

Showing Live Data in a Table, for information on setting maximum display rows.

Repository Directory, in *Advanced Server Administration*, for information on the SREE Home directory.

Visibility of Saved Worksheet Assets

When you save a Worksheet, all of its constituent assets (Data Tables, Named Groups, Variables, etc.) are saved together. However, only the asset marked as “Primary” is accessible to reports and other Worksheets. See *Setting an Asset as Primary* for information about setting the external visibility of Worksheet assets.

The asset marked “Primary” determines the appearance of the Worksheet icon in the Asset Repository. See *Using an Asset* for a list of icons.

1.2.5 Closing a Data Worksheet

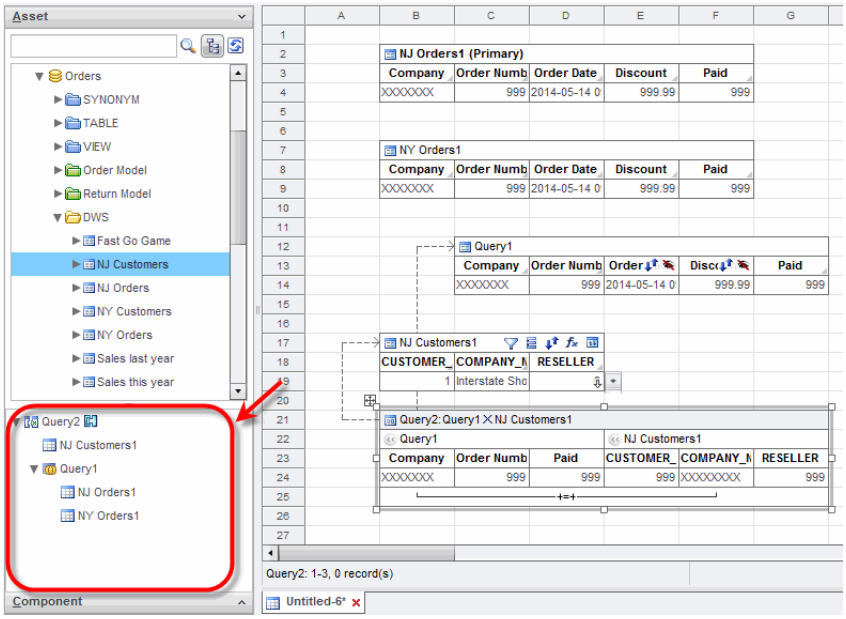
To close a Worksheet, do one of the following:

- Press Ctrl-W on the keyboard.
- Click the ‘X’ in the Worksheet tab at the bottom of the screen.

If the Worksheet contains unsaved changes, you will be prompted to save the Worksheet. See *Saving a Data Worksheet* for more information.

1.2.6 Navigating a Data Worksheet

The bottom left corner of the Visual Composer contains the *Worksheet Explorer*. The Worksheet Explorer provides a hierarchical (tree) view of all the Data Blocks in the current Worksheet. To select a particular Data Block in the Worksheet, click the appropriate node in the tree.



The hierarchy reflects the composition structure of Data Blocks in the Worksheet, with the participating (base) tables of each Composition Table listed as sub-nodes under the result.

The icons in the tree indicate the type of operation performed to obtain the resulting Data Block. Click on an icon to open the corresponding dialog

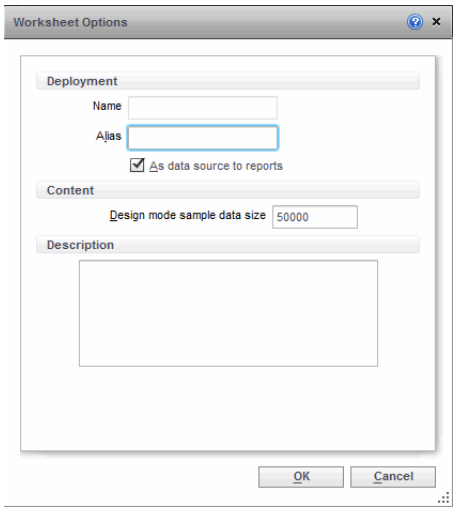
box (‘Conditions’, ‘Join Types’, etc.) and make modifications to the operation.

ICON	OBJECT/OPERATION
	*Cross Join*.
	Embedded table. (See *Creating an Embedded Table*.)
	*Inner Join*.
	*Intersect*.
	*Merge Join*.
	*Minus*.
	Mirror table. (See *Mirroring a Table*.)
	Rotated table. (See *Rotating a Table*.)
	Table used as subquery within condition. (See *Using a Subquery in a Condition*.)
	*Union*.

1.2.7 Setting Global Worksheet Options



To set global Worksheet options using Visual Composer, click the ‘Options’ button in the toolbar. In Style Studio, select the ‘Worksheet Properties’ option from the *File* menu.



There are several global options available:

- ‘As data source to reports’ specifies that the Worksheet is accessible for data binding in reports, as well as for use by Viewsheets.

- ‘Alias’ specifies a designation to be used in place of the actual Worksheet name. The alias replaces the Worksheet name in all contexts except the **Objects** tab and the ‘Export Assets’ page in Enterprise Manager. These always display the actual Worksheet name.
- ‘Description’ specifies a description for the Worksheet to appear in the Style Studio ‘Open’ dialog box and the Portal’s Ad Hoc Wizard.
- ‘Design mode sample data size’ globally limits the number of rows returned by any query or subquery for Live Preview mode, and for Viewsheet design view.

If a table in the Worksheet is composed from multiple sub-queries, each individual subquery result will be capped at the specified ‘Design mode sample data size’ maximum. This may prove efficient when the final table is itself row-limited. For example, if the final table displays only 20 rows in Live Preview mode, it may be unnecessary for the component sub-queries to each retrieve thousands of rows, and restricting these query result sets may be desirable. However, artificially limiting queries in this way can also yield an unrepresentative Live Preview result in some cases.

See Also

Using a Data Table in a Report, for details on report data binding.
Limiting the Number of Rows in a Table, for a survey of methods.
Viewing the SQL Query Plan of a Table, to inspect composition of the table query.

2 Assets: Reusable Worksheet Components

An *asset* is a reusable Worksheet component. The five different types of Worksheet assets are listed below.

ASSET	DESCRIPTION
Data Table	An array of data drawn from a data source.
Named Grouping	A reusable classification (grouping) structure.
Variable	A reusable interface for acquiring user input.

When you save an asset in the Asset Repository, you can then use that asset in multiple Worksheets and in multiple reports. This section covers the basics of working with assets.

2.1 Using an Asset

The Asset Repository tree on the left side of the Visual Composer displays all assets that are within the scope of the current Worksheet. Each asset type is shown with a different icon.

	Data Table
	Named Group
	Variable

Assets listed under the ‘Global Worksheet’ folder are accessible to all Worksheets, while assets listed under the ‘User Worksheet’ folder are accessible only to the user who originally saved those assets. (See *Controlling Access to an Asset* for information about setting access scope.)

You can use assets in both reports and Viewsheets, as well as in other Worksheets. This section explains the various methods of using assets.

2.1.1 Using a Data Table

You can use a Data Table from the Asset Repository in a Viewsheet, report, or within another Worksheet.

Using a Data Table in a Viewsheet

A Viewsheet can be based on a Worksheet. The Viewsheet will have access to *all* of the Data Tables its underlying Worksheet, regardless of whether the tables are marked as Primary.

To create a new Viewsheet based on an existing Worksheet, follow these steps:

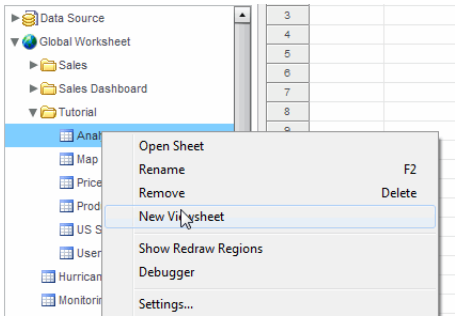


1. Click the ‘New Viewsheet’ button in the Visual Composer toolbar. This opens the ‘New Viewsheet’ dialog box.

2. In the ‘New Viewsheet’ dialog box, expand the ‘Global Worksheet’ or ‘User Worksheet’ folders, and select the Worksheet on which the Viewsheet should be based.

3. Click ‘OK’ to close the dialog box and open the new Viewsheet.

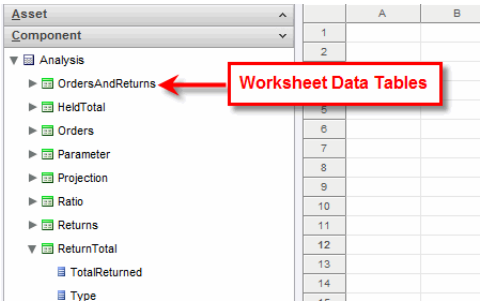
Alternatively, right-click the desired Worksheet in the Asset Repository, and select ‘New Viewsheet’ from the context menu.



To see which Data Tables of the Worksheet are accessible to an existing Viewsheet, follow these steps:

1. Open the Viewsheet in Visual Composer.

2. Expand the top node in the ‘Component’ tree. (This node has the same name as the underlying Worksheet).



The top node lists all of the Data Tables in the underlying Worksheet. Although you can insert these tables directly into the Viewsheet, it is more typical to bind the Data Tables to Viewsheet input and output controls that provide visualization and exploration capabilities.

For more information on using Worksheet data within a Viewsheet, see *Advanced Dashboard Design*.

Using a Data Table in a Report

To use a Worksheet Data Table in a report, follow the steps below.

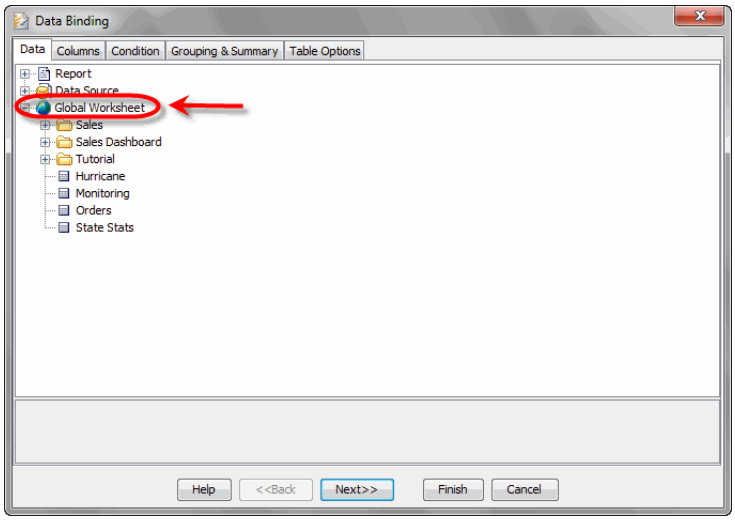


1. Click the 'Options' button in the Worksheet toolbar, and verify that 'As data source to reports' is enabled. This makes the Worksheet visible for data binding in reports.

2. In Style Studio, open the report template in which you want to access the Worksheet table.

3. Open the 'Data Binding' dialog box for the report element (Table, Chart, Section, etc.) that you want to bind to the Worksheet table. (See *Data Binding*, in *Advanced Report Design* for information.)

4. In the **Data** tab of the 'Data Binding' dialog box, expand the 'Worksheet' node on the tree.



   The 'Worksheet' node shows Data Tables belonging to the 'Global' and 'Report' scopes of the Asset Repository. Tables belonging 'User' scope are not available for binding.

5. Select the desired Data Table, and proceed with the data binding.

Note that only the table marked as 'Primary' in a Worksheet is displayed under the 'Worksheet' node of the 'Data Binding' window.

See Also

Data Binding, in *Advanced Report Design*, for details about binding.
Setting Asset Scope, for information about the different Worksheet scopes.
Setting an Asset as Primary, for details on making a Data Table the Primary asset.

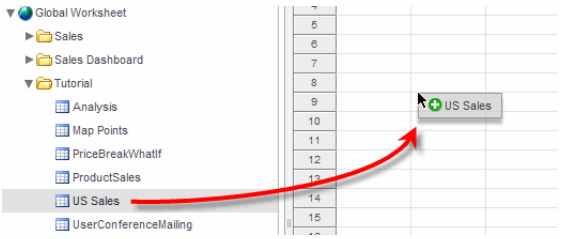
Setting Global Worksheet Options, to make Worksheets accessible to reports.

Using a Data Table within a Worksheet

To use a Data Table from the Asset Repository in a Worksheet, follow these steps:

1. Open the Worksheet in which you want to use the Data Table.
2. Select the desired Data Table asset (Worksheet) in the Asset Repository.
3. Drag the Data Table from the Asset Repository into the Worksheet.

   This adds the Data Table to the Worksheet as a new Table. Note that only the Primary Data Table of the Worksheet in the Repository is added to the new Worksheet.



4. Rename the new table as desired.

The new Data Table remains linked to the original Worksheet, and columns of the new table can therefore not be deleted. If ‘Auto Update’ is set on the new table, changes made to the original table will automatically propagate to the new table.

See Also

Setting an Asset as Primary, for information on exposing a Worksheet asset.

Renaming an Asset in the Worksheet, for information on changing a table name.

Automatically Updating an Asset, to set updating for linked assets.

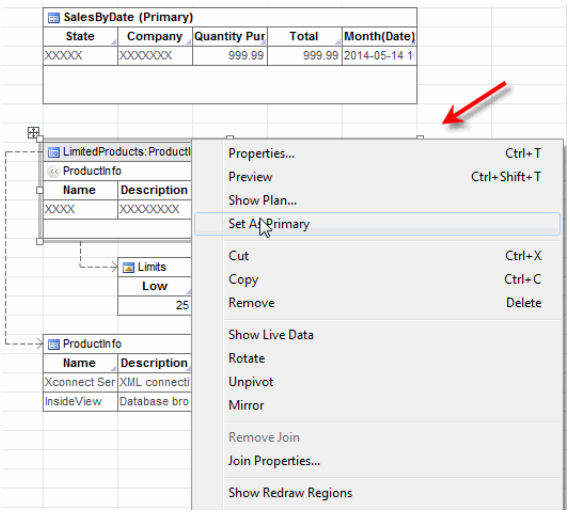
Example: Using a Data Table

The ‘ProductSales’ Worksheet (in the ‘Tutorial’ folder) contains a joined table called ‘LimitedProducts’ that returns products in a particular price range. To use this table within another Worksheet, follow the steps below:

1. In the left-side ‘Asset’ pane of Visual Composer, expand the ‘Global Worksheet’ node, and expand the ‘Tutorial’ node.

2. Double-click the ‘ProductSales’ node to open the ‘ProductSales’ Worksheet in Visual Composer.

3. Right-click the ‘LimitedProducts’ Data Block, and select ‘Set as Primary’ from the context menu. This makes the ‘LimitedProducts’ Data Block visible to other Worksheets.

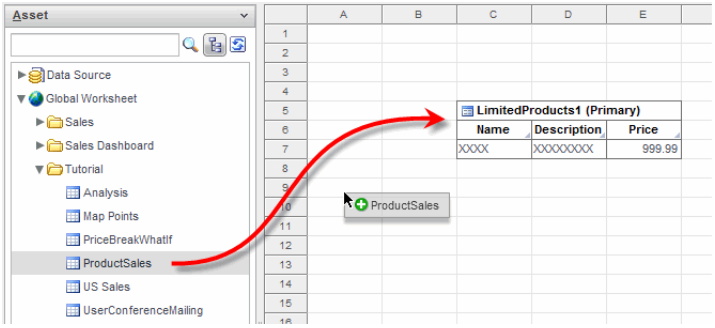




4. Save the ‘ProductSales’ Worksheet.

5. Create a new Worksheet.

6. Select the ‘‘ProductSales’ Worksheet in the ‘Global Worksheet’ tree, and drag it onto one of the empty cells in the new Worksheet. This adds the ‘LimitedProducts’ Data Block to the new Worksheet, and renames it to ‘LimitedProducts1’.



You can use the ‘LimitedProducts1’ Data Block in the new Worksheet in the same way as any other Data Block.

¢

2.1.2 Using a Named Grouping

To use a Named Grouping from the Asset Repository in a Worksheet table, follow these steps:

1. Drag the Named Grouping from the Asset Repository into the Worksheet where it will be used.

2. Select the Named Grouping from the grouping options in the ‘Group and Aggregate’ dialog box.

To use a Named Grouping from the Asset Repository in the data binding grouping of a Report, select the Named Grouping from the ‘Predefined Named Group’ menu.

See Also

Grouping and Aggregating Data for information about the ‘Group and Aggregate’ dialog box.
Data Binding, in *Advanced Report Design*, to bind report elements.
Creating a Named Grouping, to create Named Groups.

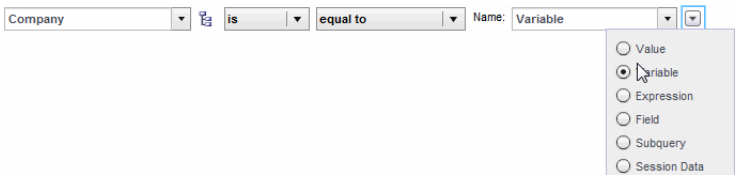
2.1.3 Using a Variable

You can use a Variable from the Asset Repository in the filtering condition of a Worksheet Data Table or Named Grouping. Follow these steps:

1. Drag the Variable from the Asset Repository into the Worksheet where it will be used.

2. Open the ‘Conditions’ dialog box for the table or Named Grouping.

3. Select the ‘Variable’ option, and choose the desired Variable from the menu.



When you bind a Variable to a table in a Worksheet (either directly through the table’s filtering condition, or indirectly through a Named Group that is linked to the table), a user will be prompted to supply the Variable value when they do one of the following things:

- Open the Worksheet in Visual Composer or Style Studio
- Save the Worksheet in Visual Composer or Style Studio
- Click the ‘Enter Parameters’ button in the Worksheet toolbar

- Open or preview a Viewsheet linked to the Worksheet (only if Viewsheet's 'Disable Prompt Parameter' option is deselected, and no default is specified for the Variable)
- Open or preview a report that uses the table linked to the Variable

See Also

Filtering, for details on setting conditions.

Setting Viewsheet Options, in *Advanced Dashboard Design*, to enable Viewsheet prompting.

Creating a Named Grouping, for details on Named Groups.

Example: Using a Variable

This example continues the example in *Defining a Variable*. Assume you have a table listing the summarized information of the sales for all states and you are interested in working with the total sales value for only one particular state at a time. This can be achieved by adding a filtering condition on the 'State' field of the table to be equal to the value of the 'State' Variable.

Instead of creating a new 'State' Variable for this Data Worksheet, you can use the existing 'State' Variable from the Asset Repository (see *Defining a Variable*).



1. Create a new Worksheet.
2. Expand the 'Data Source' node, and then the 'Orders' node.
3. Drag the 'sales by state' query onto an empty cell on the Worksheet. This creates the table 'sales by state1'.
4. Expand the 'Global Worksheet' node.

The Variable name is shown in italics to indicate that the asset is *referenced* from another Worksheet.

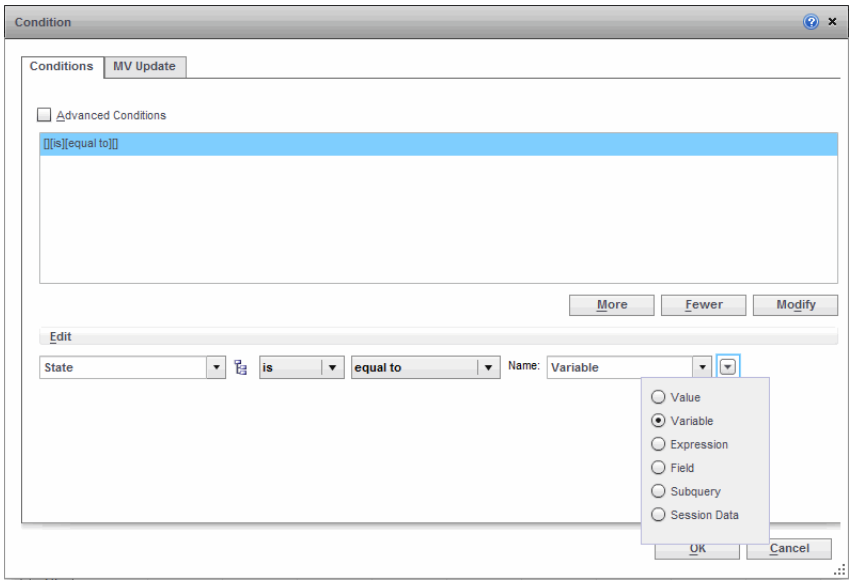
5. Select the 'State' Variable and drag it onto an empty cell in the Worksheet. This imports the variable into the Worksheet as '*state1*.'



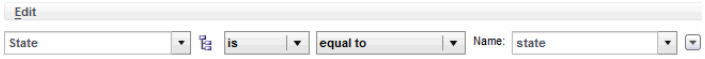


6. Click on the 'Condition' button, located in the title row of the 'sales by state' table. This opens the 'Condition' dialog.
7. Deselect the 'Advanced Conditions' box.
8. Click the 'More' button to add a new condition.
9. Select 'State' from the fields drop-down list.
10. Select "is equal to" as the condition operator.

11. For the value, specify that it is a Variable by clicking on the down arrow button to the right of the value text field and selecting ‘Variable’.

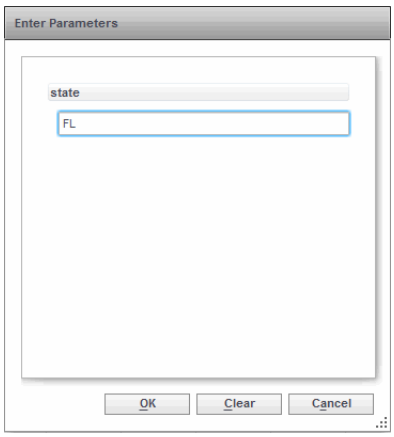


12. From the menu to the left of the down arrow, select ‘state’.



13. Click on the ‘OK’ button.

14. You will be prompted for the value of the ‘state’ Variable. Specify “FL” to be the parameter value and click on the ‘OK’ button.



15. Preview the table (see *Previewing a Table*), and note that only the sales for the state of Florida are now shown.

¢

2.2 Saving an Asset

There are two ways to save an asset: Save the Worksheet as a whole, or drag the individual asset to the repository. The following sections explain the two methods.

2.2.1 Saving an Asset by Saving the Worksheet

To save an asset (e.g., Variable) to the Asset Repository, follow these steps:

1. Mark the asset as Primary.

2. Save the Worksheet containing the asset.

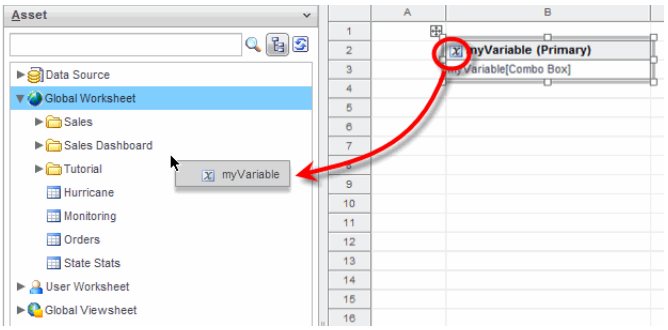
The asset marked as Primary determines the appearance of the saved Worksheet in the Repository, and how it can be used.

See Also
Setting an Asset as Primary, for details on marking the Primary asset.
Saving a Data Worksheet, for details on saving the current Worksheet.
Using an Asset for more information on asset appearance and usage.

2.2.2 Saving an Asset by Dragging to the Repository

To save an asset (e.g., Variable) to the Asset Repository, follow these steps:

1. Click the small icon in the top-left corner of the asset title bar.



2. Drag the icon over a Global Worksheet or User Worksheet folder, and release. This opens the ‘Enter Asset Name’ dialog box.

3. In the ‘Enter Asset Name’ dialog box, enter the name under which the asset should appear in the Repository, and click ‘OK’.

The asset is added to the repository in the desired folder under the specified name. See *Using an Asset* for information about how assets can be used in Worksheets.

2.2.3 Deploying an Asset

In Style Studio, you can easily deploy an asset to a repository other than the working repository (i.e., a *remote* repository) or to a JAR file for later use. See *Deploying an Asset* for more information.

2.3 Renaming an Asset

You can rename an asset inside the Worksheet that contains it. You can also rename an entire Worksheet in the Asset Repository.

2.3.1 Renaming an Asset in the Worksheet

To change the name of a table or other asset (Variable, Named Group, etc.) in a Worksheet, follow these steps:

1. Right-click the asset title row, and select ‘Properties’ from the context menu.
2. In the ‘Table Properties’ dialog box, edit the ‘Name’ field and click ‘OK’.

All of the assets in a given Worksheet must have unique names.

Example: Renaming an Asset

Say you have a table bound to the ‘sales by state’ query and you would like to name the table ‘Sales Broken Down By State.’ Follow these steps to change the name:



1. Create a new Worksheet.
2. Expand the ‘Data Source’ node, and the ‘Orders’ node.
3. Drag the ‘sales by state’ query on to an empty cell of the Worksheet.
4. Right-click on the table’s title row and select ‘Properties’ from the context menu.
5. Change the name ‘sales by state’ to ‘Sales Broken Down By State.’
6. Click ‘OK’ to finalize the change.

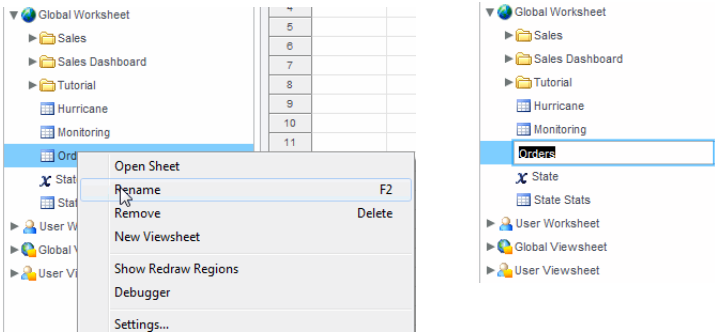


¢

2.3.2 Renaming an Asset in the Asset Repository

To change the name of a Worksheet in the Asset Repository, follow these steps:

1. Right-click the asset in the Repository, and select ‘Rename’ from the context menu.
2. Type the desired name for the asset in the provided field.
3. Click away to save the change.



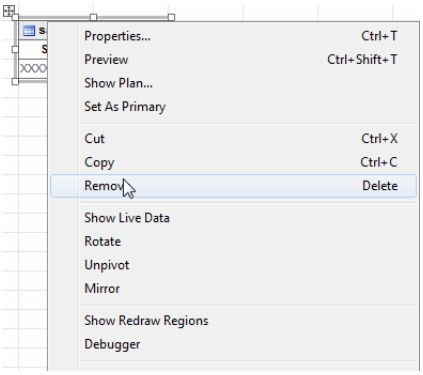
2.4 Deleting an Asset

You can delete an asset inside the Worksheet that contains it. You can also delete an entire Worksheet in the Asset Repository.

2.4.1 Deleting an Asset in the Worksheet

To delete a table or other Worksheet asset, follow these steps:

1. Click the asset title row to select the asset.
2. Right-click the asset title row to open the context menu.
3. Select ‘Remove’ from the context menu.



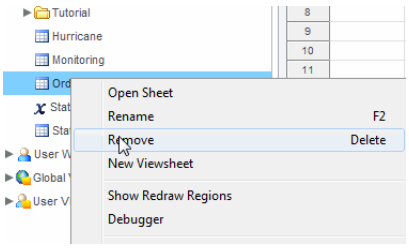
Alternatively, press the ‘Delete’ key on the keyboard to remove the assembly.

2.4.2 Deleting an Asset in the Asset Repository

To delete an asset in the Asset Repository, do one of the following things:

```
Note: If an asset is used by other reports, the asset cannot be
deleted.
```

- Select the asset name from the tree and press the Delete key on the keyboard.
- Right-click on the asset name in the tree and select the ‘Remove’ option.

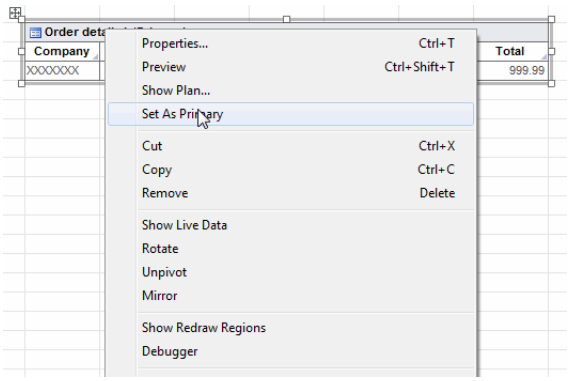


2.5 Editing a Non-Tabular Asset

To modify a non-tabular asset (Named Grouping or Variable), right-click the asset title bar, and select ‘Properties’ from the context menu. See *Creating Non-Tabular Assets* for full details about asset properties.

2.6 Setting an Asset as Primary

To mark an asset as the Primary asset, right-click on the asset’s title row, and select ‘Set as Primary’ from the context menu. The Primary asset is displayed with a bold title.



The Primary asset is the “main result” of the Worksheet. It is this Primary asset that you will access from within reports or other Worksheets.

When you save a Worksheet, the Primary asset determines the appearance of the Worksheet in the Asset Repository, and how the Worksheet can be used. See *Using an Asset* for more information about accessing Worksheet assets.

2.7 Automatically Updating an Asset

When you drag an asset from the Asset Repository into a Worksheet, the imported asset is a copy that remains linked to its own defining Worksheet. When you make changes to the original asset, the changes are propagated to all linked copies. This default behavior is called “Auto Update.”



- To disable Auto Update, click the ‘Disable Auto Update’ button in the asset title bar.



- To re-enable Auto Update, click the ‘Enable Auto Update’ button in the asset title bar.

2.8 Controlling Access to an Asset

Assets can be assigned a specific scope, which determines their accessibility to different users.

2.8.1 Setting Asset Scope

Assets can be stored under one of the following three scopes:

- **Global scope**: Assets saved under the Global scope are available to all of the users and reports in the system.
- **User scope**: Assets saved under the User scope can be accessed only by the reports and Data Worksheets that share the same user scope. User scope is only available in the Visual Composer.
- **Report scope**: Assets saved under Report scope can be accessed only by the associated report. See *Editing a Data Worksheet in Style Studio* for details.

In the Global scope only, assets can be further organized into domains, for example:

– Shared assets
– Sales domain
– Marketing domain
– R&D domain
– Support domain
– Financial domain

Each domain can be assigned different access rights. See *Component/ Object Permissions* in the *Advanced Server Administration* for more information on setting Global security for Worksheets.

2.8.2 Changing Asset Scope

To change the scope of an asset in the Asset Repository, drag it to the branch of the Repository tree that corresponds to the desired scope. The following rules apply:

- The user should have write permission to the target scope.
- If an asset is used in a report that is not owned by the current user, it cannot be moved to the User or Report scope.
- If an asset is used in more than one report, it cannot be moved to the Report scope.

Note that Report scope is only available in Style Studio, and User scope is only available in Visual Composer.

3 Creating a Data Table

You can combine data from multiple queries and different data sources into a Data Table, producing a single dataset that contains meaningful and useful information. Tables have many different roles; see *Using a Data Table* for more information.

Many Data Table operations create and maintain links between the component Data Tables, conditions, expressions, etc. These links are displayed graphically by dashed lines connecting the assets. When the configuration or value of a linked asset is changed, the impact of the change propagates to all its dependent components. This dynamic data updating makes it easy to explore the data and to perform cause and effect analysis.

3.1 Creating a Regular Data Table

This section discusses the various ways to create a regular Data Table.

3.1.1 Creating a Table from a Query or Data Model

To create a regular Data Table from a query or data model, drag one of the following assets from the Asset Repository tree into the Worksheet:

- Query
- Query column
- Data model entity
- Data model attribute

This creates a new Data Table in the Worksheet. If you drag a query column or a data model attribute into the Worksheet, then a table with just one column is created. If you drag a full query or a data model entity to the Worksheet, then a table is created which contains all the columns in the selected query or all the attributes in the selected entity.

The new Data Table displays a single data row containing the table's metadata, a representation of the table column's contents (e.g., "XXXX" for text, "9999" for integer, etc.). To view the table data, see *Showing Live Data in a Table* and *Previewing a Table*.

Example: Creating a Data Table from a Query

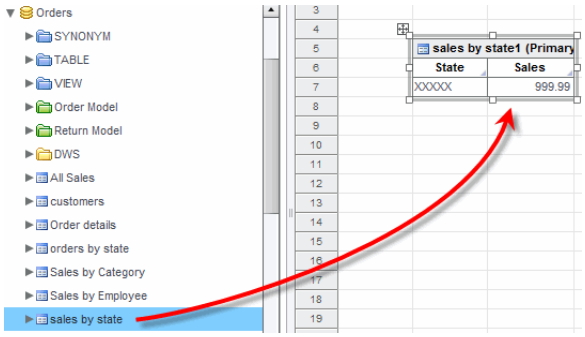


Assume that you want to create a table for the 'sales by state' query from the 'Orders' data source. Follow the steps below.

1. Create a new Worksheet by clicking on the 'New Worksheet' button.

2. Expand the 'Data Source' node, and the 'Orders' node.

3. Drag the ‘sales by state’ query from the tree to an empty cell in the Worksheet. A new table named ‘sales by state1’ is created with two columns, ‘State’ and ‘Sales’.



4. Right-click on the table title row and select ‘Preview’ from the context menu.

	sales by state1 (Prima	
2	State	Sales
3	AZ	456333.85
4	CA	1665880.05
5	CO	838303.20
6	CT	161689.50
7	FL	377534.80
8	IL	558702.80
9	MA	2126975.20
10	MD	1399917.10
11	NJ	3219337.20
12	NV	395348.70
13	NY	2441696.65
14	PA	420043.30
15	TX	2465336.95
16	WA	923016.25

¢

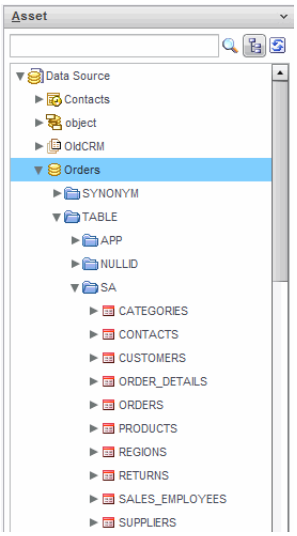
3.1.2 Creating a Table from Database Tables

In most cases, you should access data from a relational database only via *queries* and *data models*. (See the *Advanced Data Modeling* for information on creating queries and data models.) However, you can also directly access the underlying database tables.

```
Note: Whenever possible, use queries or data models to create
Worksheet assets. Only use database tables directly if this is
absolutely necessary.
```

To create a Worksheet table or column directly from a database table, follow these steps:

1. Expand the ‘TABLE’ folder in the Asset Repository tree. The database tables are now listed in the tree.



If the ‘TABLE’ folder is empty or nonexistent, your administrator may have disabled the listing of the physical fields. If the ‘TABLE’ folder does not reflect recent changes to the database schema, your administrator may need to refresh the metadata. See *Specifying Data Source Information* in *Advanced Server Administration*.

2. Drag and drop a table or table column into the Worksheet. (This is the same as adding a query or query column to the Worksheet. See *Creating a Regular Data Table* and *Adding a Table Column* for details.)

3.1.3 Creating a Table from Existing Table Columns

You can create a new Data Table by copying the columns of an existing regular Data Table. Follow the steps below:

1. Select the columns to copy. (Ctrl-click the header cells to select multiple columns. Continue holding the Ctrl key after selecting the last column.)

2. While still holding down the Ctrl key, drag the selected columns out of the table and drop them into an empty Worksheet cell.

This creates a new table containing just the selected columns. The order of the columns is the order in which they were selected.

To add columns to an existing table, see *Adding a Table Column*. To copy an entire table, see *Copying a Table*.

Example: Creating a Data Table from Columns

Assume you have a table listing the order details which include the company name, the product purchased, the description of the product, the price of the product, the quantity of that product purchased, the discount

given, and the total amount of money paid by the customer for that product in an order. you can create another table with only the information about the customer and the name of the product purchased. Follow these steps:



1. Create a new Worksheet by clicking on the ‘New Worksheet’ button.

2. Expand the ‘Data Source’ node, and the ‘Orders’ node.

3. Drag the ‘Order details’ query onto one of the empty cells on the Worksheet. This creates a new table called ‘Order details1’.

4. Ctrl-click the ‘Company’ and ‘Product’ column headers to select them.

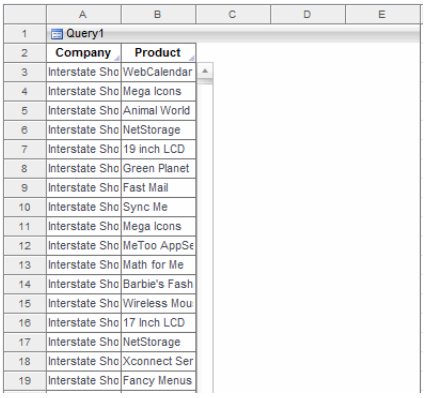
After you select both columns, you can drag either column to drag both.

5. Drag the two selected column headers out of the table and drop them onto an empty Worksheet cell.

6. A new table, ‘Query1’ is created with ‘Company’ and ‘Product’ as its columns.



7. Right-click on the title row of the ‘Query1’ table and select ‘Preview’ The resulting table shows only the ‘Company’ and ‘Product’ columns.



¢

3.2 Creating an Embedded Table

Instead of creating a regular table from a query or data model, you can create a Data Table by manually typing values into the Worksheet. Such a table is called an *Embedded Table*.

There are two ways to generate an embedded table:

- Create a new embedded table using the 'Embedded Table' option in the 'New Object' menu.
- Convert a regular Data Table into an embedded table.

After you generate an embedded table, you can use it in the same way as a regular Data Table. Embedded tables also provide a means for Viewsheets to filter the data returned by Worksheet queries, enabling straightforward *what-if* analysis.

See Also

Creating a Regular Data Table, to create a table from a query or data model.

What-If Analysis, to use embedded tables in exploratory analysis.

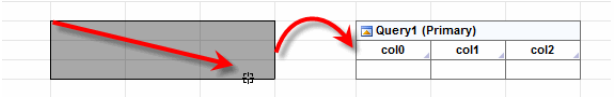
Using Input Components, in *Advanced Dashboard Design*, for information on the embedded table interface to Viewsheets.

3.2.1 Creating a New Embedded Table

To create a new embedded table, follow these steps:

1. Click the 'New Object' button, and select 'Embedded Table'. The cursor changes to a cross.

2. Click and drag the cursor across the cells that you want to compose the table. The cells are highlighted as you drag across them.



   When you release the cursor, the highlighted cells are converted into an embedded table. The top row of cells becomes the title row of the table, the second-to-top row becomes the column header row of the table, and any remaining rows of highlighted cells become the data rows of the table.

3. To edit a column header, double-click on the header and enter the desired text.

4. To edit a data cell, click in the cell and enter text directly. You can enter both numeric and non-numeric values. (See *Changing the Column Type in an Embedded Table*.)

Example: Creating a New Embedded Table

Assume that you want to create a table of values to be used in a formula for some other table on the Worksheet. (Refer to *What-If Analysis* for such an application of the embedded table.) In the following example you will create an embedded table with three rows and three columns of data.



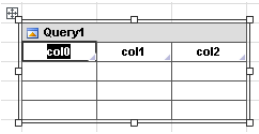
1. Create a new Worksheet by clicking on the 'New Worksheet' button.



2. Click on the 'New Object' button, and select the 'Embedded Table' option. The cursor changes to a cross.

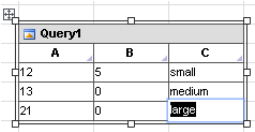
3. Drag across the Worksheet to highlight five rows and three columns. When you release the mouse, the selected range of the empty grid cells becomes part of the new table called 'Query1'; the first row of the selected range becomes the header row, and the second row becomes the column header row.

4. All the cells of this newly created table are editable. Double-click on the 'col0' header cell. The cell switches to editing mode, and you can now type in the cell to change the text.



5. Change the column names for 'col0', 'col1' and 'col2' to 'A', 'B' and 'C', respectively.

6. Enter any numeric values in the nine data cells. Note that both numeric and non-numeric values can be entered in the table.



¢

See Also

Adding or Removing a Row from an Embedded Table, to change the table length.

3.2.2 Converting a Regular Table to an Embedded Table

To convert a regular Data Table to an embedded table, follow these steps:

1. In the table title bar, click the ‘Change View’ button. Select ‘Live Preview’ from the menu. This will populate the table with data.

2. Right-click the title bar of the table, and select ‘Convert to embedded Table’.

This creates a new table with the name of ‘Query’. Once the table is converted into an embedded table, it *cannot* be reverted back to a regular table.

Example: Converting to an Embedded Table

In the following example, the ‘Sales by Category’ query is converted to an editable embedded table.

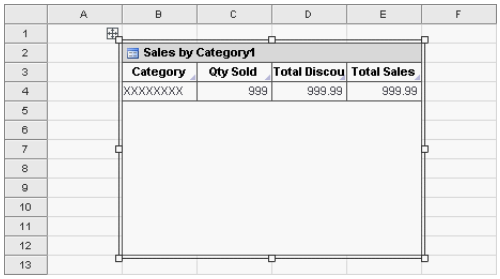


1. Create a new Worksheet by clicking on the ‘New Worksheet’ button.

2. Expand the ‘Data Source’ node, and then the ‘Orders’ node.

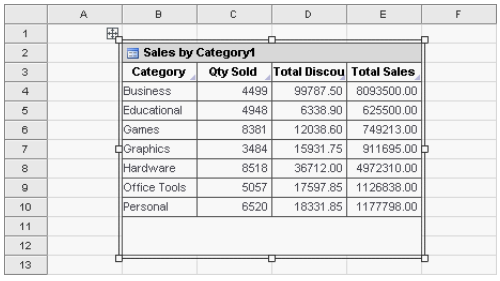
3. Drag the ‘Sales by Category’ node from the tree to an empty cell in the Worksheet. A new table named ‘Sales by Category1’ is created.

4. Expand the table vertically so that it occupies a total of 11 rows in the Worksheet.

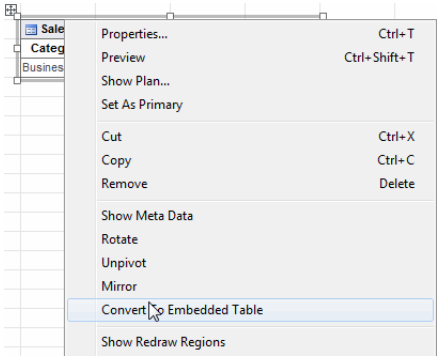




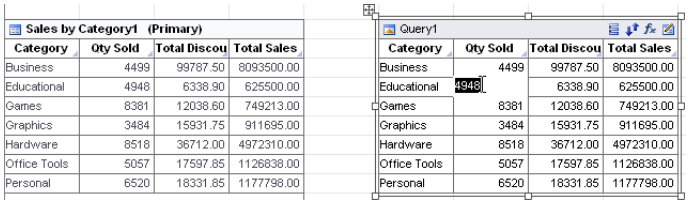
5. Click the ‘Change View’ button in the table title bar, and select ‘Live Preview’ from the menu.



6. Right-click on the title row and select ‘Convert to Embedded Table’. A new table will be created with a name of ‘Query1’. Expand the number of rows so that all rows are visible.



7. Now, click on any of the cells of the ‘Query1’ table. Notice that all of the cells in this table are now editable.



Sales by Category1 (Primary)

Category	Qty Sold	Total Discou	Total Sales
Business	4499	99787.50	8093500.00
Educational	4948	6338.90	625500.00
Games	8381	12038.60	749213.00
Graphics	3484	15931.75	911695.00
Hardware	8518	36712.00	4972310.00
Office Tools	5057	17597.85	1126838.00
Personal	6520	18331.85	1177798.00

Query1

Category	Qty Sold	Total Discou	Total Sales
Business	4499	99787.50	8093500.00
Educational	4948	6338.90	625500.00
Games	8381	12038.60	749213.00
Graphics	3484	15931.75	911695.00
Hardware	8518	36712.00	4972310.00
Office Tools	5057	17597.85	1126838.00
Personal	6520	18331.85	1177798.00

¢

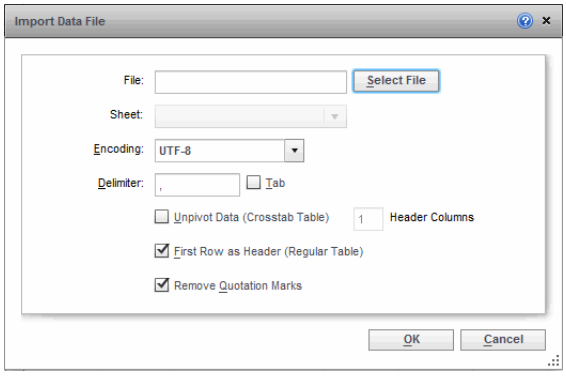
3.2.3 Importing Data Into an Embedded Table

To import data from a delimited text file into an embedded table, follow the steps below:

The size of the embedded table will change to match the number of rows and columns in the imported data.

1. Create a new embedded table, or select an existing embedded table. (Existing data and column names will be overwritten by the imported data.)

2. Right-click the title bar of the embedded table, and select ‘Import Data File’ from the context menu. This opens the ‘Import Data File’ dialog box.



3. Click ‘Select File’, and choose the desired text file. (By default, only files with the extension “.csv” are shown. To view all file types, enter “*.*” in the ‘File Name’ field, and press Enter.)

4. From the ‘Encoding’ menu, select the encoding of the text file.

5. In the ‘Delimiter’ field, enter the character that is used in the text file to separate entries. The default delimiter character is a comma. For tab-delimited files, select the ‘Tab’ option.

6. Adjust any of the following optional settings to modify the way the file data is interpreted:

   a. Select ‘First Row as Header (Regular Table)’ to use the entries in the first row of the text file as the column header text. Otherwise, default column names are used (“col0,” “col1,” etc.).

   b. (Optional) Select ‘Unpivot Data (Crosstab Table)’ if the data in the file is in crosstab table form, and you wish to convert the crosstab table into a regular table within the Worksheet. This option will convert the column header row into a “dimension” column and convert the crosstab cells into a “measure” column. If the crosstab table in the file contains multiple levels of row headers, enter the number of levels in the ‘Header Columns’ field.

   You can only unpivot a crosstab that has a single level of column headers.

   c. (Optional) Select ‘Remove Quotation Marks’ to strip quotation marks out of the text file upon import.

The delimited text is imported into the embedded table, and the table resizes itself to the correct number of columns.

See Also

Unpivoting a Table, to unpivot an arbitrary Worksheet crosstab.
Creating a New Embedded Table, for instructions on adding a new table.

Text Data Source, in *Advanced Data Modeling*, for more extensive text capabilities.

3.3 Adding a Table Column

You can expand an existing table in the Data Worksheet by adding additional columns. This allows you to gather data from multiple entities into a single table. You can also create new columns that contain complex calculations.

This section discusses various methods for adding table columns.

3.3.1 Adding a Column to a Regular Table

To add a new column to a regular table in the Worksheet (from the same query or data model), follow these steps:

1. In the asset tree, select the query column or a model attribute that you want to add to the existing table.

2. Drag the query column or model attribute over the column header row in the existing table. (A green bar shows the location in the table where the new column will be placed.)

3. Drop the column into the table at the desired location.

See Also

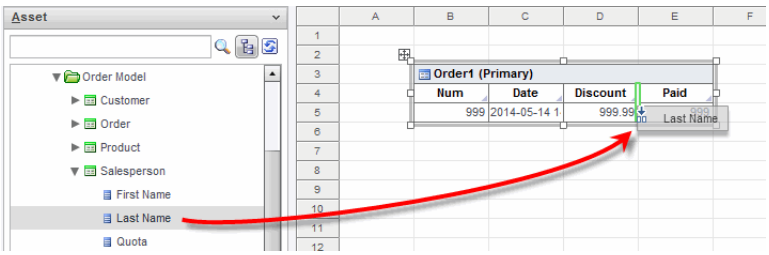
Joining Tables, to integrate columns from different queries or models.
Merging Tables, to merge tables that share the same data model or query.

Example: Adding a Column to a Regular Table

In the following example, you add a single attribute (salesperson last name) as a new column in an existing table.

1. Drag the ‘Order Model’ > ‘Order’ entity to an empty cell on the Worksheet. This creates a new ‘Order1’ table.

2. Drag the ‘Last Name’ attribute from ‘Order Model’ > ‘Salesperson’ entity over any column header in the existing ‘Order1’ table. (A green bar indicates where the new column will be placed.)



3. Release the mouse button when the green bar appears in the location you wish to place the new column. The column is then added to the table.

¢

3.3.2 Adding a Column to an Embedded Table

To add a new column to an embedded table, right-click a column header. Choose ‘Insert Column’ to insert a blank column to the left of the column you clicked, or choose ‘Append Column’ to append a blank column to the right of the column you clicked. All cells in the new column can be edited.

See Also
Adding or Removing a Row from an Embedded Table, to change the table length.

3.3.3 Adding or Removing a Row from an Embedded Table

To **add a new row** to an embedded table, follow these steps:

1. Deselect the table. (Click the Viewsheet grid away from the table.)

2. Right-click a table row, and choose ‘Insert Row’ or ‘Append Row’ from the context menu. This inserts a blank row above the row you clicked or appends a blank row below the row you clicked, respectively.

To **remove an existing row** from an embedded table, follow these steps:

1. Deselect the table. (Click the Viewsheet grid away from the table.)

2. Right-click on the table row you want to delete, and choose ‘Remove Row’ from the context menu. This removes the row you clicked, and shifts subsequent rows upward in the table.

3.3.4 Creating an Expression/Formula Column

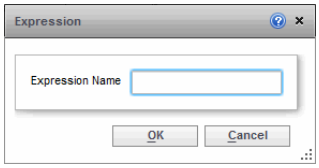
An expression column, or formula column, is a column whose data is generated from a script (expression) by processing existing data in the table. The expression can use either SQL or JavaScript syntax, and can reference any other columns within the Data Table.

To add an expression column to a table, follow these steps:



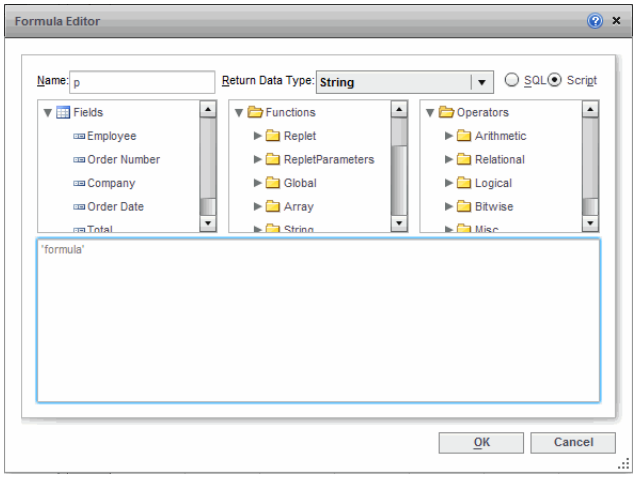
1. Click the ‘Expression’ button (labeled ‘*fx*’) located in the title row of the Data Table. This opens the ‘Expression’ dialog box.

2. Enter a name for the new column in the dialog box, and click ‘OK’.



This adds the formula column to the right side of the table, and opens the Formula Editor.

3. In the Formula Editor, click the ‘SQL’ button to enter an expression using SQL syntax, or press the ‘Script’ button to enter an expression using JavaScript syntax. Select the appropriate data type from the ‘Return Data Type’ menu.



4. Enter the desired formula expression in the bottom text field.

   JavaScript expression columns can make full use of the built-in scripting functions in the script library. You can reference the value of a Viewsheet input component, Viewsheet parameter, report parameter, or Worksheet variable by using the component name or variable name as a *parameter*. For example, if a Viewsheet input control has name ‘RadioButton1’, you can reference the input’s value within a Worksheet expression by using syntax “`parameter.RadioButton1`”.

   See *Input Components* in *Advanced Dashboard Design* for more information.

5. Click ‘OK’ to exit the Formula Editor. The data in the new column will reflect the results of the specified expression.

Example: Creating a Formula Column

Consider the ‘Sales by Category’ query, which lists the total revenue generated from the sales of different categories of products. It also has

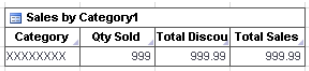
information about the total discount given for each product category. Using these two pieces of information, you can calculate the 'Revenue' generated by each product category. Follow these steps to perform the calculation.



1. Create a new Worksheet by clicking on the 'New Worksheet' button.

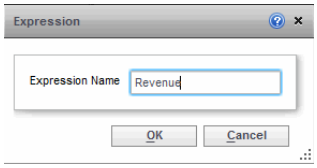
2. Expand the 'Data Source' node, and the 'Orders' node.

3. Drag the 'Sales by Category' query from the tree onto an empty cell in the Worksheet. A new data block, 'Sales by Category1', is created.



4. Click on the 'Expression' button, located on the title row of the 'Sales by Category1' table. This opens the 'Expression' dialog box.

5. Enter "Revenue" as the name of the new expression column.



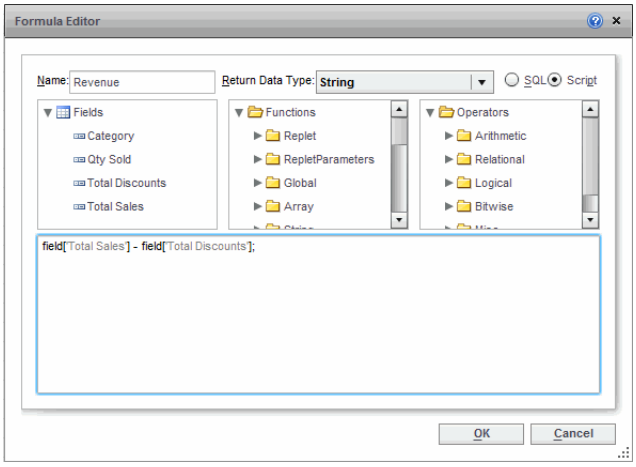
6. Click 'OK'. This appends a new 'Revenue' column to the right side of the Data Table, and opens the Formula Editor.

7. In the 'Return Data Type' menu, select the 'Float' option.

8. Enter the following formula in the text area:

You can click on a field name in the left panel to add it to the expression area.

```
field['Total Sales'] - field['Total Discounts'];
```



(Optional) To edit the expression again at a later time, click the '*fx*' button in the expression column header (not the table title). This reopens the Formula Editor.

Refer to Appendix B:*Accessing Table Cells in Script* for information on how to reference the current table cell, row, and column in the expressions.

9. Preview the 'Sales by Category1' table.

	A	B	C	D	E
1	Sales by Category1				
2	Category	Qty Sold	Total Discou	Total Sales	Revenue
3	Business	4499	99787.50	8093500.00	7993712.50
4	Educational	4948	6338.90	625500.00	619161.10
5	Games	8381	12038.60	749213.00	737174.40
6	Graphics	3484	15931.75	911695.00	895763.25
7	Hardware	8518	36712.00	4972310.00	4935598.00
8	Office Tools	5057	17597.85	1126838.00	1109240.15
9	Personal	6520	18331.85	1177798.00	1159466.15

¢

See Also

Editing an Expression Column, to modify the formula of an expression column.

Appendix B:*Accessing Table Cells in Script*, to reference table data in expressions.

Appendix JS:*General JavaScript Functions*, for further information.

3.3.5 Automatically Substituting JavaScript For SQL

There are certain conditions that may prevent a SQL expression from being executed properly (for example, if your particular database does not honor the Data Block 'Maximum Rows' setting or the administrative 'Query Maximum Row Count' setting). To prevent the expression script from failing in such cases, you can provide a fall-back JavaScript expression to be evaluated when the SQL script cannot be executed.

To include a fall-back JavaScript expression, simply enclose the desired JavaScript expression within `/*script [...] script*/` tags. For example:

```
CASE WHEN field['Returns.Return Total'] IS NULL THEN
field['Orders.Order Total']
ELSE field['Orders.Order Total']-field['Returns.Return
Total']
END

/*script
if(field['Returns.Return Total'] == null) {
   field['Orders.Order Total']
}
else {
   field['Orders.Order Total']-field['Returns.Return
Total']
}
script*/
```

In this case, the JavaScript version of the expression within the `/*script [...] script*/` tags will be executed *only if* the SQL version cannot be executed.

3.3.6 Creating a Numeric Range Column

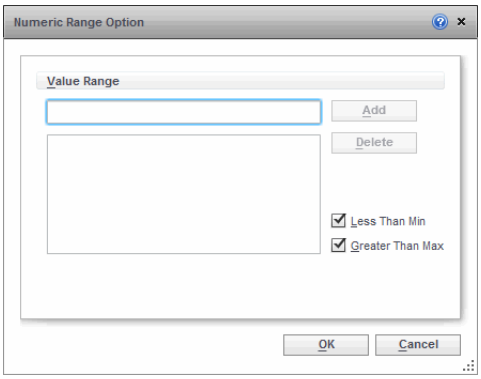
A numeric range column groups numeric data into a predefined set of bins or ranges, for example:

– Less than 5
– 5 to 10
– 10 to 15
– 15 to 20
– Greater than 20

You can create a range column for any numeric column in a Data Table. To create a range column, follow these steps:

1. Right-click the header of the column for which you want to create a range column, and select the ‘New Range Column’ option from the context menu. This opens the ‘Value Range Name’ dialog box

2. Enter a name for the new range column, and click ‘OK’. This opens the ‘Numeric Range Option’ dialog box. Here you can specify the different benchmarks defining the range.



To customize a benchmark label, double-click the benchmark value. to enable editing.

3. Enter a benchmark value into the ‘Value Range’ field, and click the ‘Add’ button. Repeat until all desired benchmarks have been entered.

4. Select the ‘Less Than Min’ checkbox to create a bin for all values that fall below the minimum benchmark. If you do not select this option, those values are classified as “Others.”

5. Select the ‘Greater Than Max’ checkbox to create a bin for all values that fall above the maximum benchmark. If you do not select this option, those values are classified as “Others.”

6. Click the ‘OK’ button to close the dialog box.

See Also

Creating a Named Grouping, for another way of partitioning column data.
Creating a Date Range Column for partitioning date information.

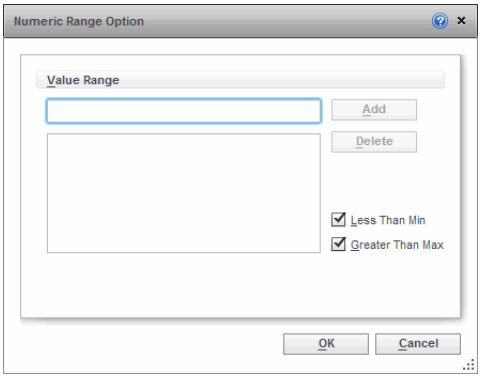
Example: Creating a Numeric Range Column

Consider the ‘Sales by Category’ query, which lists the total amount generated from the sales of different categories of products. You can define a range column for the ‘Total Sales’ field, which places each amount into a predefined range (bin).

1. Create a new Worksheet and drag the ‘Sales by Category’ query into a vacant cell.

2. Right click the ‘Total Sales’ column, and select the ‘New Range Column’ option. This opens the ‘Value Range Name’ dialog box.

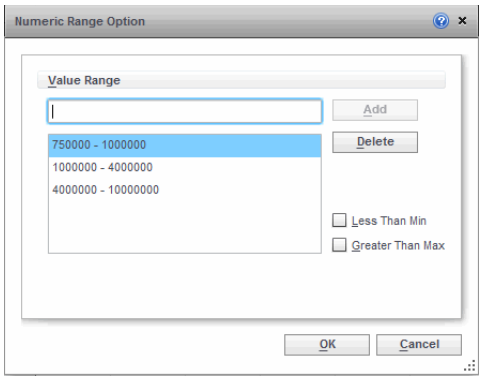
3. Enter the name “Revenue Range” and click ‘OK’. This creates the new column and opens the ‘Numeric Range Option’ dialog box. Here you can specify the different benchmarks in your range.



4. Deselect the ‘Less Than Min’ and ‘Greater Than Max’ options.

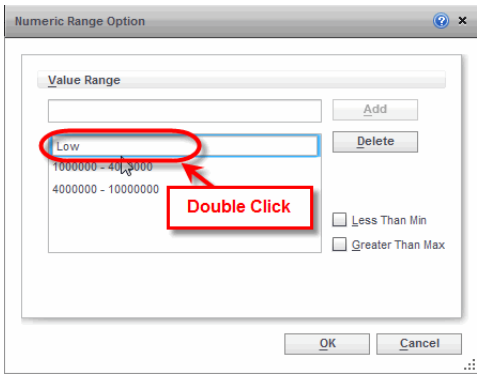
5. Enter the first benchmark value, 750000, into the ‘Value Range’ field, and click the ‘Add’ button.

6. Enter the next three benchmarks in turn, clicking ‘Add’ after each one: 1000000, 4000000, 10000000.



This creates the three ranges: (1) 750,000-1,000,000, (2) 1,000,000-4,000,000, and (3) 4,000,000-10,000,000.

7. Double-click the ‘750000-1000000’ label and enter the new label “Low”.



8. Repeat the above step to relabel the 1,000,000-4,000,000 range as “Medium” and the 4,000,000-10,000,000 range as “High”.

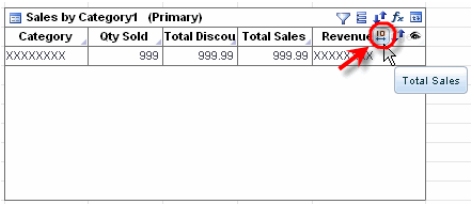
9. Click the ‘OK’ button to close the dialog box.

10. Preview the table data. Note that each cell in the ‘Total Sales’ column has a corresponding range in the ‘Revenue Range’ column. The values that lie outside the specified range are labeled ‘Others’.

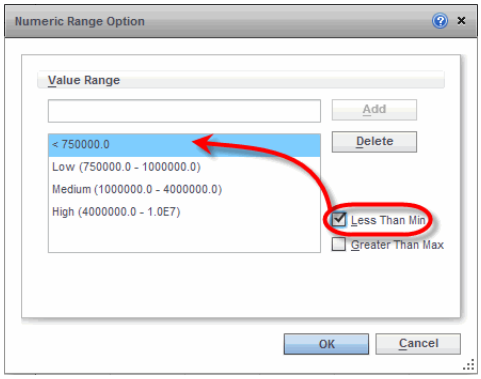
	A	B	C	D	E
1	Sales by Category1				
2	Category	Qty Sold	Total Discou	Total Sales	Revenue Ra
3	Business	4499	99787.50	8093500.00	High
4	Educational	4948	6338.90	625500.00	Others
5	Games	8381	12038.60	749213.00	Others
6	Graphics	3484	15931.75	911695.00	Low
7	Hardware	8518	36712.00	4972310.00	High
8	Office Tools	5057	17597.85	1126838.00	Medium
9	Personal	6520	18331.85	1177798.00	Medium

11. (Optional) Instead of defining a fixed minimum and maximum value (750,000-10,000,000), you can keep the numeric range open ended. Follow the steps below:

a. Open the ‘Numeric Range Option’ dialog box by clicking the icon in the column header (see *Editing a Range Column*).



b. In the dialog box, select the ‘Less Than Min’ option, and click the ‘OK’ button.



c. Preview the Data Table and note that the values outside the specified range (formerly labelled ‘Others’) are now labelled ‘<750000.00’.

¢

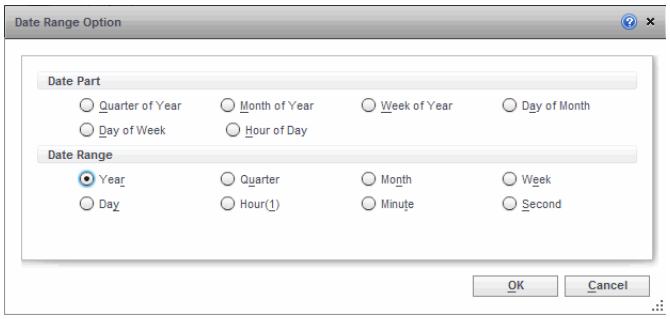
3.3.7 Creating a Date Range Column

A date range column groups dates into a fixed set of date bins. You can create a range column for any date column in a Data Table.

To create a range column, follow these steps:

1. Right-click the header of the column for which you want to create a range column, and select the ‘New Range Column’ option from the context menu. This opens the ‘Date Range Name’ dialog box.

2. Enter a name for the new column in the ‘Date Range Name’ dialog box, and click ‘OK’. This opens the ‘Date Range Option’ dialog box.



3. Select the desired partitioning (year, month, quarter, etc.), and click the ‘OK’ button to close the dialog box.

See *Creating a Numeric Range Column* for doing a similar partitioning on numeric columns.

Available Date Ranges

There are five date ranges available for the date range column:

- ‘Year’: A number in the range 0000-9999
- ‘Quarter’: A number in the range 1-4
- ‘Month’: A number in the range 0-11
- ‘Day of Week’: A number in the range 1-7
- ‘Day of Month’: A number in the range 1-31

3.4 Editing a Table Column

This section describes various ways to change the column properties of a table. Column sorting options are discussed in the *Sorting* section of *Manipulating Tabular Data*.

3.4.1 Renaming a Column

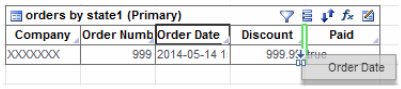
```
Warning: If there is a Viewsheet that uses on this Worksheet
column, renaming the column will cause the Viewsheet to lose its
reference.
```

To change a column name, double-click the column header and enter the desired text.

3.4.2 Reordering a Column

To change the position of a column, click on the column header cell, and drag the column to the desired location in the table. A green bar shows the location where the column will be inserted.

In the figure below, the ‘Order Date’ column is positioned between the ‘Discount’ and ‘Paid’ columns.

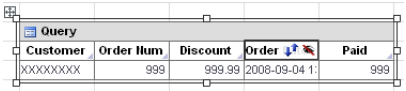


3.4.3 Showing and Hiding a Column

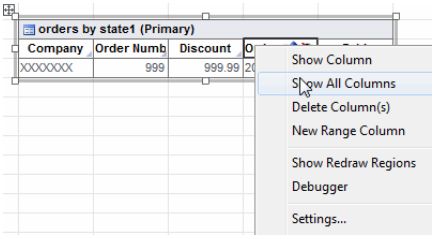


To change the visibility of a column (show or hide), click the ‘visibility’ button next to the column name in the column header cell.

In the figure below, the ‘Order Date’ column is hidden.



To show all data block columns, right-click on any column header and choose ‘Show All Columns’ from the context menu.



3.4.4 Changing the Column Type in an Embedded Table

By default, all manually created columns in an embedded table are of type ‘String’. To change the type of an embedded table column, follow these steps:

1. Right-click the column header, and select ‘Column Type’ from the context menu. This opens the ‘Column Type’ dialog box.

2. Select the desired data type from the ‘Type’ menu. If you select a numeric or date type, a secondary ‘Parse Format’ menu appears below.

This allows the Embedded Table to convert the data to a standard representation.

3. (Optional) In the ‘Parse Format’ menu, select the format in which you have entered your data. For example, if you have entered dates in the form “1/13/1995,” select or type “MM/d/yyyy” as the parse format.

4. Click ‘OK’ to close the ‘Column Type’ dialog box.

After you have specified a data type for a column of an Embedded Table, any additional data you enter into the column must match this specified type.

Example: Changing the Column Type

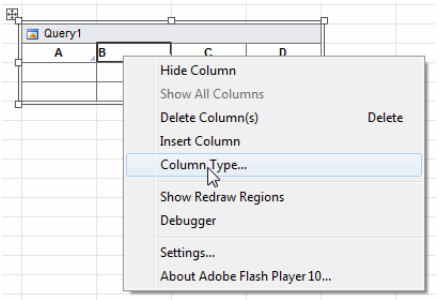
In the following example, you will change the data type of an embedded table column from ‘String’ to ‘Integer’:



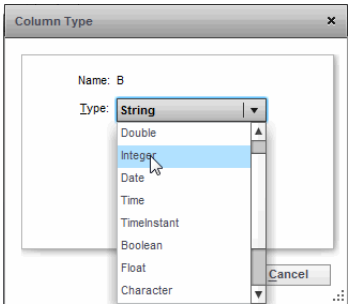
1. Create a new Worksheet.

2. Create a new embedded table with four rows and four columns. See *Creating an Embedded Table* for instructions on how to create an embedded table.

3. Right-click a header cell and select ‘Column Type’ from the context menu. This opens the ‘Column Type’ dialog box.



4. Select ‘Integer’ from the ‘Type’ menu and click ‘OK’.



¢

See Also

Date and Time Formats, in *Advanced Report Design*, for more information on formats.

Format Tab, in *Advanced Dashboard Design*, for more information on Viewsheet formatting.

3.4.5 Editing an Expression Column

To change the formula of an expression column, follow these steps:

1. Click the column header to select it.
2.  Click the 'Formula' button in the column header. This opens the Formula Editor.
3. Make the desired changes to the formula, and click 'OK' to close the Formula Editor.

See *Creating an Expression/Formula Column* for more information about using the Formula Editor.

3.4.6 Editing a Range Column

To edit a date range column or numeric range column, follow these steps:

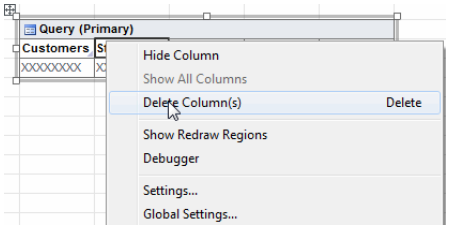
1. Select the column header.
2.  Click the Date Range or Numeric Range icon in the header.

This opens the 'Numeric Range Option' or 'Date Range Option' dialog box. See *Creating a Numeric Range Column* and *Creating a Date Range Column* for more details about using these dialog boxes.

3.5 Deleting a Table Column

To delete a column or multiple columns from a table, follow these steps:

1. Click the header of the first column to delete. This selects the column.
2. Ctrl-click the headers of any other columns to delete. These columns are also selected.
3. Right-click on any of the selected column headers, and choose 'Delete column(s)' from the context menu. The columns are deleted from the table.



Deleted columns are removed from the query. To hide rather than delete a column, see *Showing and Hiding a Column*. To add a column, see *Adding a Table Column*.

3.6 Copying a Table

To eliminate repetitive work, an existing table may be copied to create a new one. To copy and paste a table, follow these steps:

1. Right-click the title row of the table to copy, and select 'Copy' from the right-click menu. This copies the table to the clipboard.

2. Right-click on an empty cell in the Data Worksheet where you would like to paste the copy. Select 'Paste' from the right-click menu to paste the clipboard copy into the Worksheet.

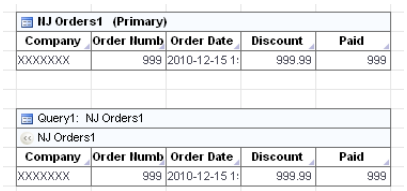
A new table is created in the Worksheet at that location. Although the newly created table is a copy of the existing table, the two tables are completely independent of each other. Therefore, changes made to one table do not affect the other table.

To create a table copy that remains linked to the original table, see *Mirroring a Table*.

3.7 Mirroring a Table

A mirrored table looks just like a copied table (see *Copying a Table*), but remains linked to the original table. This link is one-way only: Changes made to the original table are propagated to the mirrored table, but changes made to the mirrored table are not propagated to the original table.

To create a mirrored table, right-click the title row of the table to mirror, and select 'Mirror' from the right-click menu.



Original Table

Mirrored Table

The new mirrored table is created, and a graphical link to the original table is drawn to indicate the dependency.

A mirrored table is useful when you want to manipulate data in a table without affecting dependent assets. For example, if a table is being used as a component of a Composition Table, manipulation of the table's data would propagate to the dependent Composition Table as well. To prevent this propagation, you can mirror the original table and then work with the mirrored copy instead of the original.

Like a Composition Table, a mirrored table has a Hierarchical mode and an Editable mode; see *Editing a Composition Table* for more information.

3.8 Table Operations

This section explains the basic operations you can perform on tables.

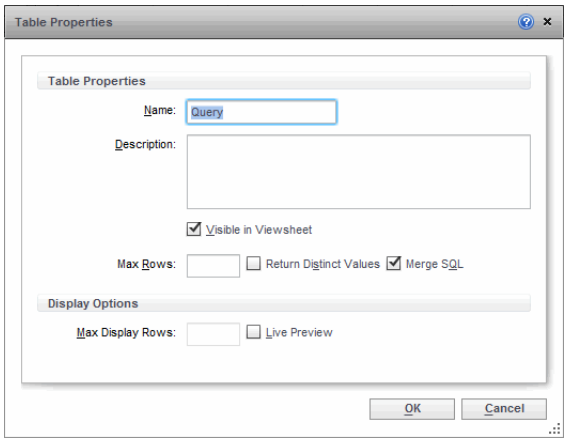
3.8.1 Setting Table Query and Display Properties

You can change various properties of a Data Table, such the table description, maximum number of rows retrieved and displayed, and the handling of duplicate rows.

To change table properties, follow these steps:

1. Right-click the title row, and select 'Properties' from the context menu. This opens the 'Table Properties' dialog box.

2. Change the desired properties, and click 'OK' to exit the dialog box.



The table properties are described below.

PROPERTY	DESCRIPTION
Description	The description text is displayed as a tooltip when the mouse hovers over the Table icon in the table's title row, and when the table is selected in Style Studio. (See *Using a Data Table* for information about binding tables.)
Max Rows	The maximum number of rows to retrieve from the database at runtime and to display in table preview. This is also the maximum number shown in Live Preview mode when 'Max Rows' is less than 'Max Display Rows'. (See *Previewing a Table* and *Showing Live Data in a Table*.)
Return Distinct Values	Display only distinct rows in the table. (Remove duplicate rows.)
Merge SQL	When selected, the Worksheet attempts to generate the table by forming a single SQL query, thus delegating all of the data manipulations to the database. If this option is not selected, or if the unified SQL statement cannot be formed, the table is generated in post-processing. (See *Viewing the SQL Query Plan of a Table* for information on viewing a table's query.) In certain cases, you can improve Viewsheet performance by disabling the 'Merge SQL' option.
Live Preview	Display live data in the table, rather than meta-data. When displaying live data, it is recommended that you set the 'Max Rows' property to prevent the query from returning a very large number of records. Live Preview can also be enabled from the 'Change View' button on the table title bar. See *Showing Live Data in a Table*.
Max Display Rows	Maximum number of table rows to retrieve and display in Live Preview mode. (See *Showing Live Data in a Table*.) This property does not affect the number of records retrieved from the database at runtime or during preview. If 'Max Rows' specifies a smaller value, the 'Max Rows' value is used as the Live Preview maximum.

3.8.2 Viewing the SQL Query Plan of a Table

To view the SQL statements that a table will attempt to execute (the query plan), right-click the title bar, and select 'Show Plan' from the context menu. This opens the 'Query Plan' dialog box.

The 'Query Plan' dialog box displays the queries that will be sent to the database at runtime. Additionally, it shows the data operations that the Worksheet will perform in post-processing, after the data is retrieved from the database. The plan also shows the source information for parameterized conditions.

By default, the Worksheet attempts to create a single SQL query that fully generates the table's data. This is typically the most efficient approach because it allows the database to perform all the needed data operations

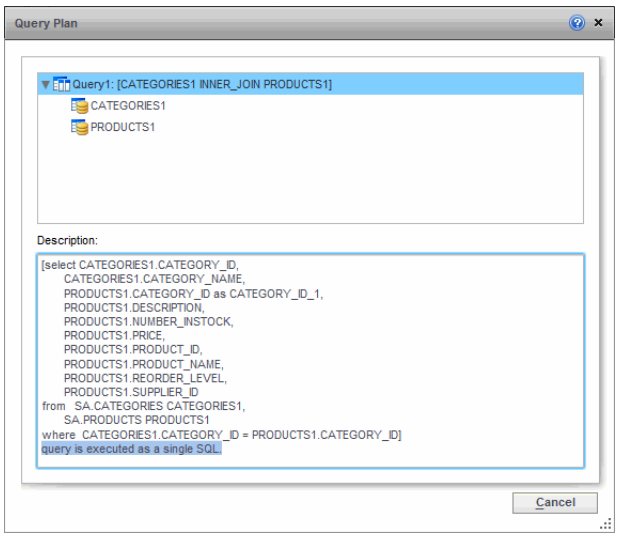
(joins, filtering, etc.). However, you can override this behavior for a particular table by deselecting the 'Merge SQL' option in the 'Table Properties' dialog box, and in some cases this may improve performance. See *Setting Table Query and Display Properties* for more information.

Example: Viewing the SQL Query Plan

This example illustrates how you can control the SQL query that a table sends to the database.

1. Expand the 'TABLE' > 'SA' node of the 'Orders' data source.
2. Drag the 'PRODUCTS' table to an empty location on the Worksheet.
3. Drag the 'CATEGORIES' table to an empty location on the Worksheet.
4. Drag the 'CATEGORY_ID' column header from the 'CATEGORIES' table onto the 'CATEGORY_ID' column header of the 'PRODUCTS' table, and release when the 'join' symbol appears.

   This creates a new table called 'Query' that contains the inner join between the two tables based on the 'CATEGORY_ID' columns. (See *Inner Join* for more details.)
5. Right-click the title bar of the 'Query' table, and select 'Show Plan'. This displays the 'Query Plan' dialog box.



The 'Query Plan' dialog box displays the query that the table will send to the database. The statement below the query indicates that this query (including the instructions to join the two tables) will be sent to the

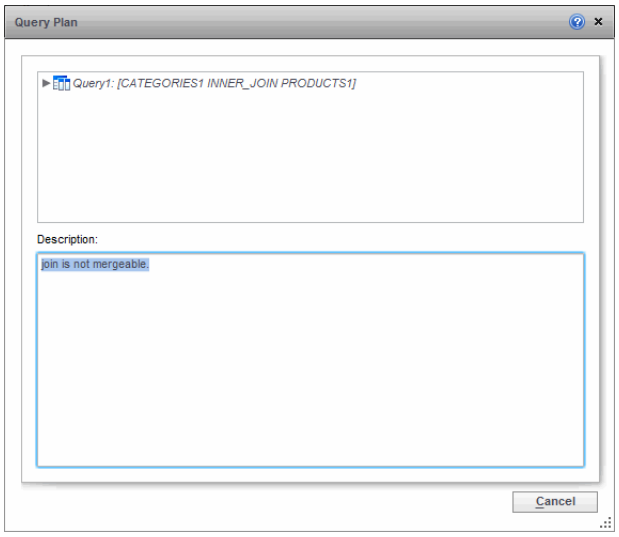
database as a single SQL statement, which is also indicated by the single icon in the top panel.

To override the default single-query behavior, follow the steps below:

1. Right-click the title row of the ‘Query’ table again, and select ‘Properties’ from the context menu. This opens the ‘Table Properties’ dialog box.

2. Deselect ‘Merge SQL’ and click ‘OK’ to exit the dialog box.

3. Right-click the table title bar, and select ‘Show Plan’ again.



The query plan is again displayed, but the top panel now shows an expandable hierarchy of queries. The ‘PRODUCTS’ and ‘CATEGORIES’ queries are each individually executed as single queries, but the inner join that generates the final ‘Query’ table is performed by the Worksheet in post-processing.

¢

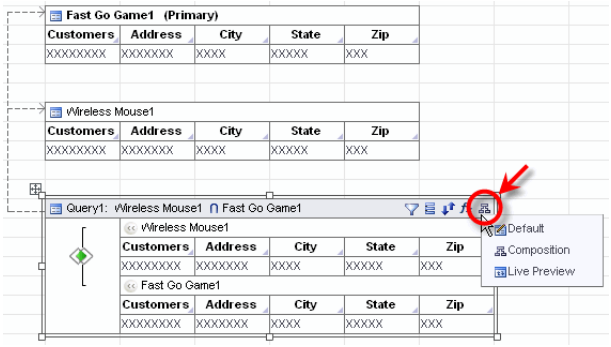
3.8.3 Editing a Composition Table

Joined Tables and Concatenated Tables are considered *Composition Tables*. A Composition Table, as well as a Mirrored Table, has two modes: *Composition view* and *Default view*.

- **Composition view** shows the exact composition of the table. It displays graphical links between the tables and columns involved in the composition, and indicates the type of concatenation or join being used. You cannot edit a table’s columns when the table is in Composition view.

- **Default view** allows you to edit column attributes, such as sorting order, visibility, etc. When a Composition Table is in Default view, it looks similar to a regular Data Table. All of the table's composition information, such as join links, concatenation types, constituent tables, etc., is hidden from view.

To toggle a Composition Table to Composition view, click the 'Change View' button in the table title bar, and select 'Composition' from the menu. To toggle a Composition Table to Default view, click the 'Change View' button in the table title bar, and select 'Default' from the menu.

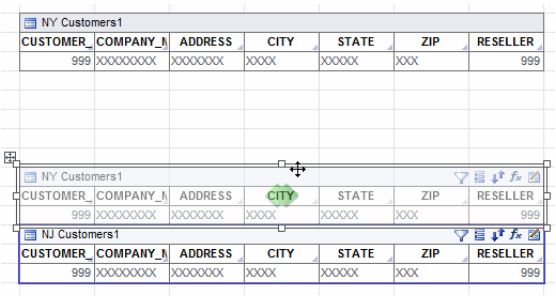


Example: Editing a Composition Table

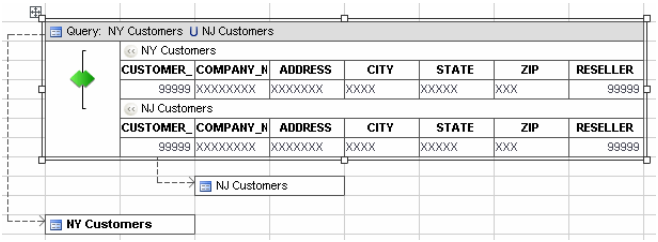
Assume that you have a Composition Table which is a union of two tables, one displaying information about customers in the state of NY and the other displaying information about the customers in the state of NJ. Both of these tables have a column named 'customer_id' which you would like to hide. They also both have a column named 'company_name' which you would like renamed to 'Company'.

To create this Composition Table, and then hide and rename columns in the Default view, follow the steps below:

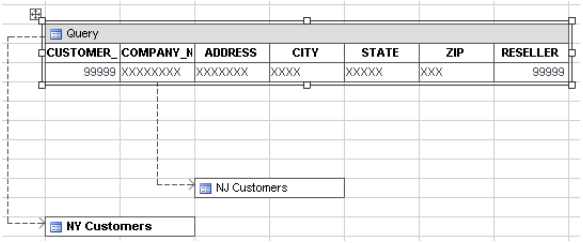
1. Create a new Worksheet.
2. Expand the 'Data Source' node, 'Orders' node, and 'DWS' node.
3. Drag the 'NY Customers' query on to one of the empty cells in the Worksheet. This creates a new table, named 'NY Customers1'.
4. Drag the 'NJ Customers' query on to one of the empty cells in the Worksheet. This creates a new table, named 'NJ Customers1'.
5. Drag the 'NY Customers1' table until it is directly above or below the 'NJ Customers1' table. When the 'union' symbol appears, release the table to perform the concatenation.



A new Concatenated Table, ‘Query1’ is created. Notice that the ‘Query1’ table is in the ‘Hierarchical’ view since the ‘union’ symbol is visible on the left of the table, indicating its type, and the table’s title row lists all of the tables participating in the union operation.

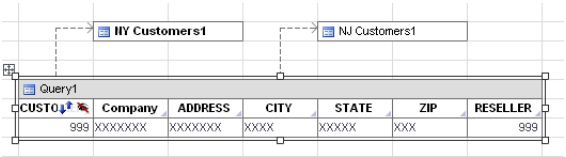


6. Press the ‘Change View’ button in the table title bar, and select ‘Default’ from the menu.

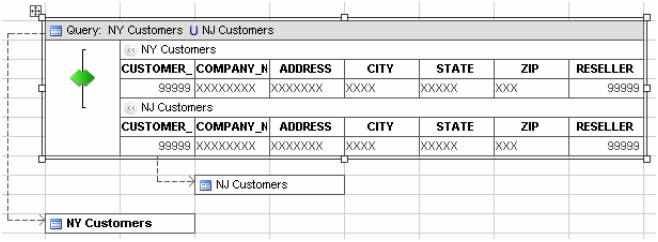


7. Select the ‘CUSTOMER_ID’ cell and click on the ‘Visibility’ button to hide it.

8. Double-click the ‘COMPANY_NAME’ cell, and change the name to ‘Company’.



9. Switch back to ‘Composition’ view by pressing the ‘Change View’ button in the table title bar. Select ‘Composition’ from the menu.



Notice that the column name change does not appear in the Composition view. This is because the Composition view displays the base table information and the column name has not been changed for the base tables.

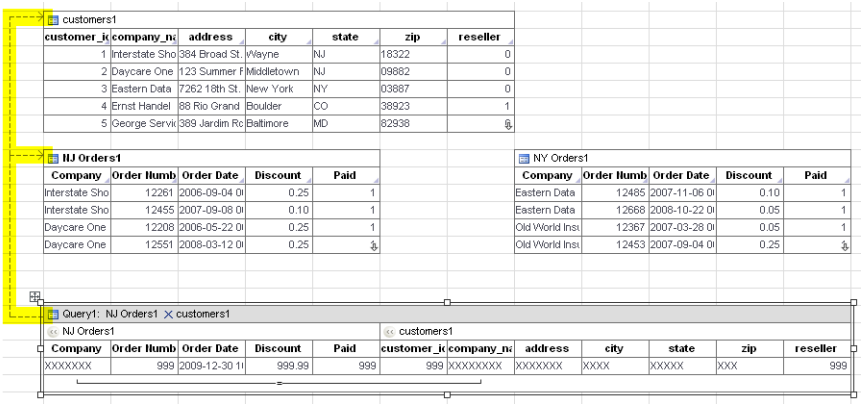
10. Preview the table. Notice that the ‘CUTOMER_ID’ field is hidden and the ‘COMPANY_NAME’ column header has been renamed to ‘Company’.

	A	B	C	D	E	F
1	Query2					
2	Company	ADDRESS	CITY	STATE	ZIP	RESELLER
3	Eastern Data	7262 18th St.	New York	NY	03887	0
4	Old World Insı	1230 Hoes La	New York	NY	08854	0
5	FISGA Corp	21 Doger Sou	New York	NY	28122	0
6	High Society I	1248 Broadw	New York	NY	27834	0
7	Interstate Sho	384 Broad St.	Wayne	NJ	18322	0
8	Daycare One	123 Summer I	Middletown	NJ	09882	0
9	Scheggie Ser	839 Via Gaty	Camden	NJ	12334	0
10	Software Spe	300 Great Sw	Newark	NJ	23445	1
11	Rutgers Bank	283 Davidson	New Brunsw	NJ	08854	0
12	Pharma Resei	38 Commercia	Plainfield	NJ	08273	0

¢

3.8.4 Changing the Source of a Composition Table

When you select a Composition Table (Joined Table or Concatenated Table), the Worksheet displays a set of arrows that indicate the table’s dependencies. For example, when you click on a Joined Table, the lines and arrows indicate the two Data Blocks are joined to produce the selected table.



In some cases, you may need to change the source of a Composition Table *after* you have already created the table. For example, when you originally created the Composition Table, you might have used a source Data Block (e.g., Embedded Table) containing some prototype or placeholder data. Later, when the production data is available, you need to replace this prototype Data Block with the Data Block containing real data.

To change the source of a Composition Table from one Data Block to another, follow the steps below:

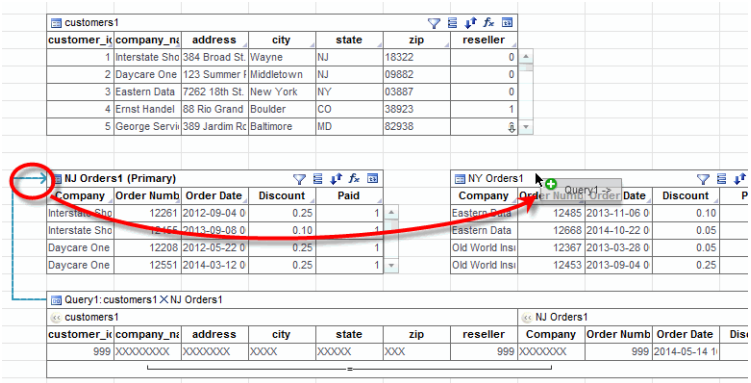
`Note: The new Data Block must be compatible with the Data Block it replaces. See below.`

1. Add the *new* (replacement) source Data Block to the Worksheet.

2. Click the Composition Table to select it. This displays the lines and arrows that show the Composition dependencies.

3. Click the arrow that connects the Composition Table to the *old* source Data Block (the Data Block you wish to replace). This highlights the arrow.

4. Drag and drop the arrow over the new (replacement) Data Block.



This rebinds the Composition Table to the new (replacement) Data Block.

To preserve the existing composition operation with the replacement Data Block, the replacement Data Block must be compatible with old Data Block in the following manner:

- For a Joined Table, the name of the original join column must match a column name in the replacement Data Block in order to preserve the join. Otherwise, the resultant join type will default to a cross-join until you define a new relationship.
- For a Concatenated Table, the number of columns and the data types in the replacement Data Block must match the old Data Block.

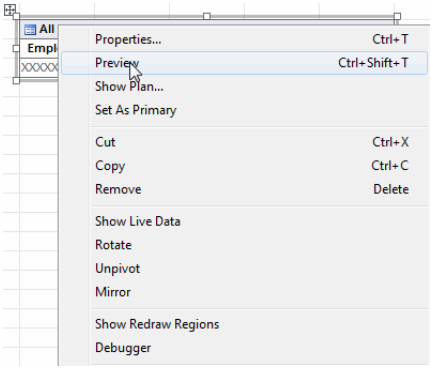
See Also

Changing the Join Type, to change the existing join operator.

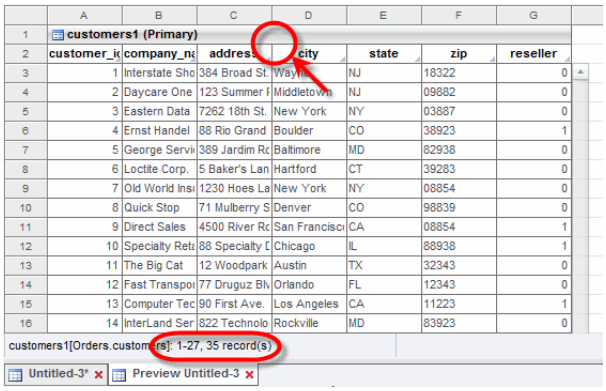
Changing the Concatenation Type, to change the existing concatenation operator.

3.8.5 Previewing a Table

To preview a table, right-click the table title bar, and select 'Preview' from the context menu. This opens the Preview tab, which displays all visible columns of the table.



You can only preview one table at a time. In the table preview tab, click the table title bar to see table information, including the record numbers being displayed. Click outside the table to see information about the Worksheet.



See Also

Showing and Hiding a Column, to limit the number of columns displayed.
Setting Table Query and Display Properties, to limit the number of rows retrieved.
Showing Live Data in a Table, to display tabular data within the Worksheet itself.

3.8.6 Showing Live Data in a Table

A new table by default displays only metadata. However, the table can also display “live data” drawn from the database. This often provides a useful alternative to Preview.

For a large table, before entering Live Preview mode, you should set the ‘Max Display Rows’ in the ‘Table Properties’ dialog box to a reasonable value. Note that for Live Preview mode, filtering conditions are only applied to the sample data retrieved from the database, which is limited by the ‘Max Rows’ and ‘Max Display Rows’ settings (as well as by the global ‘Design mode sample data size’ and administrator-set limits). Therefore, Live Preview may not accurately represent the results of table filtering.

To view live data in a table, follow these steps:

1. Click the ‘Change View’ button in the table title bar, and select ‘Live Preview’.

2. Drag the bottom border of the table downward to display more rows.

3. To scroll the table, move the mouse over the arrow symbol in the last column.



This displays the scroll bar, which you can use to scroll the data. The indices of the rows currently displayed in the table are shown in the information bar at the bottom of the screen.

To toggle the table back to Meta Data mode, press the ‘Change View’ button in the table title bar and select ‘Default’ from the menu. You can also toggle Live Preview mode from the ‘Table Properties’ dialog box.

See Also

Creating a Regular Data Table, for information about creating a new table.
Previewing a Table, for information about displaying the full table data.
Setting Table Query and Display Properties, for details about ‘Table Properties’.
Displaying the Length of a Table, for methods of determining table length.

3.8.7 Limiting the Number of Rows in a Table

There are several ways to limit the number of rows (records) presented in a table:

- To limit the number of rows retrieved and displayed by a particular table in Live Preview mode only, set the ‘Max Display Rows’ property in the table ‘Table Properties’ dialog box.
- To limit the number of rows retrieved from the database at runtime (and displayed in Preview and Live Preview modes), set the ‘Max Rows’ property in the table ‘Table Properties’ dialog box.
- To globally limit the number of rows returned by *all* queries and sub-queries, set the ‘Design mode sample data size’ in the ‘Worksheet Options’ dialog box.
- To retrieve only the rows of a table matching some specified criteria, set a filtering condition on the table.
- To retrieve only the rows of a table that have some specified relationship to the data of another table, create a joined table.

See *Tailoring Server Performance* in the *Advanced Server Administration* for other methods of limiting query size and duration.

See Also

Setting Table Query and Display Properties, for details about ‘Table Properties’.

Setting Global Worksheet Options, for details on ‘Design mode sample data size’.

Tailoring Server Performance, in *Advanced Server Administration*, for global limits.

Filtering, for information on how to set a filtering condition on a table.

Joining Tables, for limiting rows displayed by comparison with another table.

3.8.8 Displaying the Length of a Table

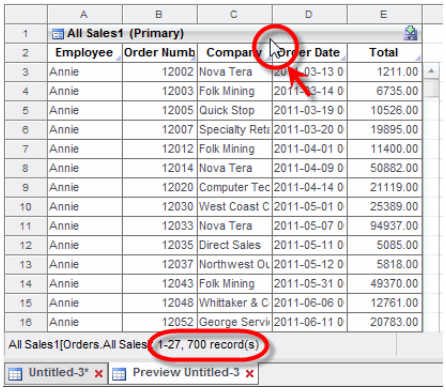
The length of a table is the number of records that the table retrieves from the database. You can check this length in two ways, by using Live Preview mode or by using Preview.

Displaying Table Length in Preview

To check the length of a table using Preview, follow these steps:

1. Select the table, and click ‘Preview’ in the toolbar. This opens the ‘Preview’ tab.

2. Click on the table title bar. The information bar at the bottom of the page shows the number of records in the table.



Displaying Table Length in Live Preview Mode

To check the length of a table using Live Preview, follow these steps:

1. Press the ‘Change View’ button in the table title bar, and select ‘Live Preview’ from the menu.

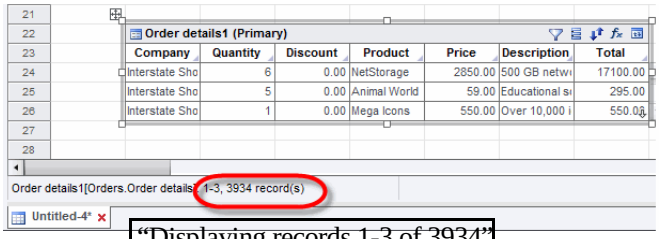
2. Open the ‘Table Properties’ dialog box.

3. Make sure the ‘Max Rows’ field is empty. (This ensures that there is no artificial limit on the number of records retrieved. However, limits can also be set by an administrator.)

4. Click ‘OK’ to close the ‘Table Properties’ dialog box.

5. Click the table title bar to select the table.

6. Consult the information bar at the bottom of the window (just above the tab).



“Displaying records 1-3 of 3934”

The information bar displays the total number of records retrieved, as well as the record numbers currently being displayed. If the information display reads “maximum exceeded,” this means that Live Preview mode is unable to display the full table. In this case, use the Preview method to assess the table length.

See Also

Showing Live Data in a Table, for information on Live Preview mode.

Setting Table Query and Display Properties, for details about ‘Table Properties’.

3.8.9 Rotating a Table

To rotate a table, right-click on the table’s title row and select ‘Rotate’ from the context menu.

This creates a rotated copy of the table, which is linked back to the original table. In the rotated copy, each row of the original table becomes a column, and each column of the original table becomes a row. The first column of the original table is the column header row of the rotated table, and the column header row of the original table is the first column of the rotated table.

3.8.10 Unpivoting a Table

In some cases it is useful to *unpivot* a table from crosstab form into normal “flat” form. This operation converts the column headers into an additional ‘Dimension’ column.

To unpivot a crosstab, follow the steps below:

1. Right-click the table title bar, and select ‘Unpivot’ from the context menu. This opens the ‘Unpivot Data’ dialog box.

2. In the ‘Levels of Row Headers’ field, enter the number of columns containing row headers.

3. Press ‘OK’ to close the dialog box.

This creates a new table which contains the same data from the original crosstab, but in flattened form (i.e., with column headers converted to an independent ‘Dimension’ column). The unpivoted table remains linked to the original crosstab so that changes to data in the crosstab are automatically propagated to the new table.

Example: Unpivoting a Crosstab

To understand how to unpivot a crosstab, first create a crosstab by following the steps below:



1. Create a new Worksheet by clicking on the ‘New Worksheet’ button.

2. Expand the ‘Data Source’ node, and the ‘Orders’ node.

3. Drag the ‘All Sales’ query from the tree to an empty cell in the Worksheet. This creates a new table named ‘All Sales1’. You will use this table to create a crosstab.



4. Press the ‘Group and Aggregate’ button in the table’s title bar. This opens the ‘Group and Aggregate’ dialog box.

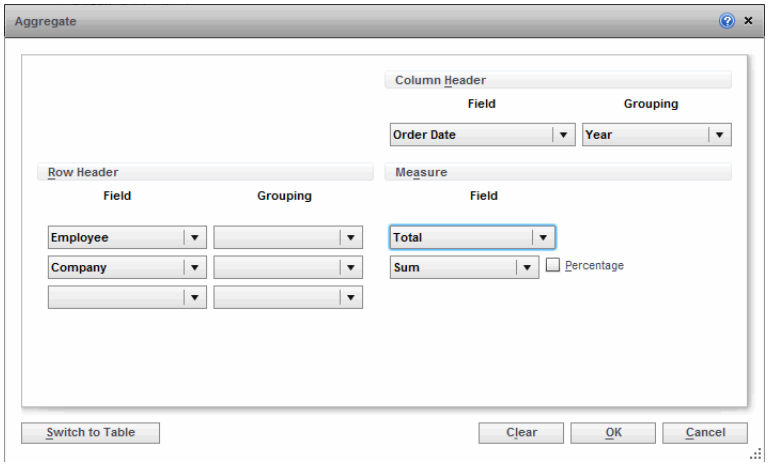
5. Press the ‘Switch to Crosstab’ button at the bottom of the dialog box. This updates the dialog box to display the crosstab binding interface.

6. In the ‘Row Header’ region, select the following two fields: ‘Employee’ and ‘Company’.

7. In the ‘Column Header’ region, select the ‘Order Date’ field.

8. In the menu next to the ‘Order Date’ field, select ‘Year’.

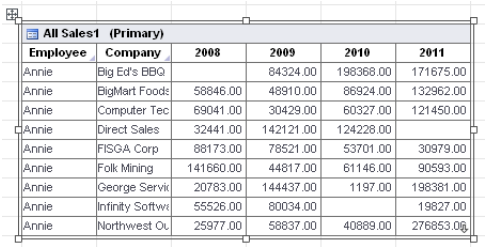
9. In the ‘Measure’ region, select the ‘Total’ field.



10. Press ‘OK’ to close the dialog box. This creates a new crosstab in the Data Worksheet.

11. In the crosstab title bar, click the ‘Change View’ button, and select ‘Live Preview’ from the menu.

12. Drag the bottom border of the table to expand the table.



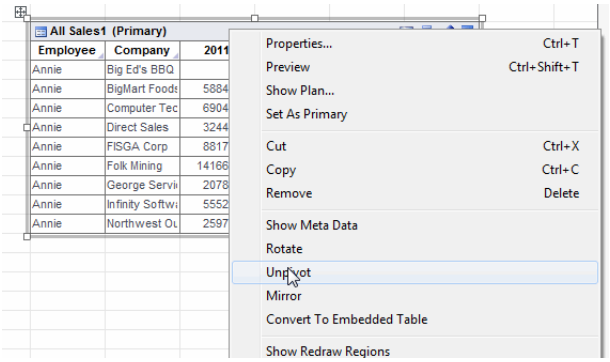
All Sales1 (Primary)

Employee	Company	2008	2009	2010	2011
Annie	Big Ed's BBQ		84324.00	198368.00	171675.00
Annie	BigMart Foods	58846.00	48910.00	86924.00	132962.00
Annie	Computer Tec	69041.00	30429.00	60327.00	121450.00
Annie	Direct Sales	32441.00	142121.00	124228.00	
Annie	FISGA Corp	88173.00	78521.00	53701.00	30979.00
Annie	Folk Mining	141660.00	44817.00	61146.00	90593.00
Annie	George Servic	20783.00	144437.00	1197.00	198381.00
Annie	Infinity Softwa	55526.00	80034.00		19827.00
Annie	Northwest Ou	25977.00	58837.00	40889.00	276853.00

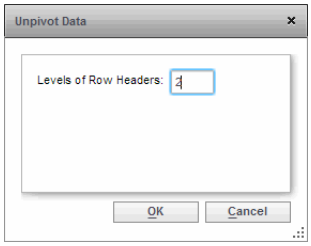
This crosstab has two levels of row headers (‘Employee’ and ‘Company’), and one level of column headers (‘Order Date’, broken out by year: 2008, 2009, 2010, 2011).

You will now *unpivot* the crosstab to create a flat table that contains the same data. Follow the steps below:

1. Right-click the crosstab title bar, and select ‘Unpivot’ from the context menu. This opens the ‘Unpivot Data’ dialog box.



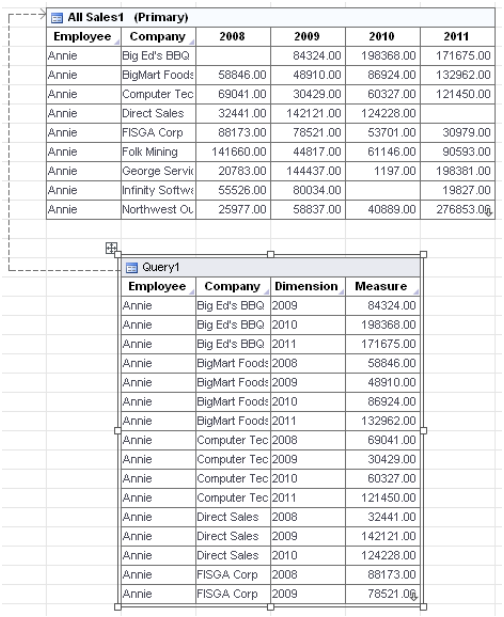
2. In the 'Unpivot Data' dialog box, enter a value of "2" for the 'Levels of row headers' value. (The two levels are 'Employee' and 'Company'.)



3. Press 'OK' to close the dialog box. This creates a new table containing the unpivoted (flattened) data.

4. In the table title bar, click the 'Change View' button, and select 'Live Preview' from the menu.

5. Drag the bottom border of the table to expand the table.



All Sales1 (Primary)

Employee	Company	2008	2009	2010	2011
Annie	Big Ed's BBQ		84324.00	198368.00	171675.00
Annie	BigMart Foods	58846.00	48910.00	86924.00	132962.00
Annie	Computer Tec	69041.00	30429.00	60327.00	121450.00
Annie	Direct Sales	32441.00	142121.00	124228.00	
Annie	FISGA Corp	88173.00	78521.00	53701.00	30979.00
Annie	Folk Mining	141660.00	44817.00	61146.00	90593.00
Annie	George Servic	20783.00	144437.00	1197.00	198381.00
Annie	Infinity Softwa	55526.00	80034.00		19827.00
Annie	Northwest Ou	25977.00	58837.00	40889.00	276853.00

Query1

Employee	Company	Dimension	Measure
Annie	Big Ed's BBQ	2009	84324.00
Annie	Big Ed's BBQ	2010	198368.00
Annie	Big Ed's BBQ	2011	171675.00
Annie	BigMart Foods	2008	58846.00
Annie	BigMart Foods	2009	48910.00
Annie	BigMart Foods	2010	86924.00
Annie	BigMart Foods	2011	132962.00
Annie	Computer Tec	2008	69041.00
Annie	Computer Tec	2009	30429.00
Annie	Computer Tec	2010	60327.00
Annie	Computer Tec	2011	121450.00
Annie	Direct Sales	2008	32441.00
Annie	Direct Sales	2009	142121.00
Annie	Direct Sales	2010	124228.00
Annie	FISGA Corp	2008	88173.00
Annie	FISGA Corp	2009	78521.00

Observe that the unpivoting operation transforms the values of the row header in the crosstab (‘Order Date’) into a new column called ‘Dimension’. The new unpivoted table retains the row headers of the original crosstab in the first two columns (‘Employee’ and ‘Company’), but now repeats each combination of ‘Employee’ and ‘Company’ for each value of ‘Order Date’.

The data presented in the unpivoted table is exactly the same as the data presented in the original crosstab. However, the flattened version may be more conducive to certain data operations such as trending.

¢

See Also

Importing Data Into an Embedded Table, for information on unpivoting a crosstab table during import.

4 Combining Data Tables

One of the Data Worksheet's most powerful capabilities is combining tables from different data sources into unified Data Blocks. This section explores the different ways that Data Tables can be combined.

4.1 Concatenating Tables

A *Concatenated Table* is a table generated by one of the following set operations: *Union*, *Intersect*, and *Minus*. These set operations are called *concatenations*. Candidate tables for concatenation should have the same number of columns, and corresponding columns of each table should have the same data type. A Concatenated Table is a type of Composition Table. See *Editing a Composition Table* for information about other properties of Composition Tables.

You can only concatenate two tables at a time, but any Concatenated Table can be used in further concatenations. To concatenate multiple base tables, first concatenate two of the base tables, and then concatenate the resulting Concatenated Table with the third base table, and so on.

4.1.1 Creating a Concatenated Table

To create a Concatenated Table, follow these steps:

1. Select two tables to concatenate. (Ctrl-click the table title rows to select them.)

   The order in which you select the tables may influence the result. See the *Minus*, *Union*, and *Intersect* sections for more details.



2. Click the 'Concatenate Table' button in the Worksheet toolbar and select the desired concatenation: 'Union', 'Intersect', or 'Minus'.

See *Changing the Concatenation Type* for information about switching the concatenation type of an existing Concatenated Table.

4.1.2 Union

By default, the union of two tables contains all of the *unique* rows from the two tables. The first table you select is the table whose distinct rows will appear first in the result. By default, no duplicate rows are retained in the union table. For information on how to preserve duplicate rows in the union table, see *Changing the Concatenation Type*.

For information on how to create a union table, see *Creating a Concatenated Table* or the example below.

Example: Union

Say you have two queries: One returning a list of customers who have purchased more than 100 pieces of ‘Wireless Mouse’ and the other returning a list of customers who have purchased more than 100 pieces of ‘Fast Go Game’. If you desire to have both these pieces of information provided as one entity, you can write another SQL query that is a union of the two above mentioned queries. However, the necessity of writing a new query can be eliminated by performing the union of the two tables, each bound to one of the two queries.

Follow the steps below to concatenate two tables in order to see the contact information of all customers who have purchased more than 100 pieces of either ‘Wireless Mouse’ or ‘Fast Go Game’ in one table.

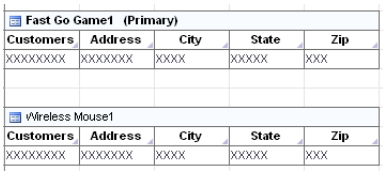


1. Create a new Worksheet by clicking on the ‘New Worksheet’ button.

2. Expand the ‘Data Source’ node, the ‘Orders’ node, and the ‘DWS’ node.

3. Drag the ‘Fast Go Game’ node from the tree to an empty cell in the Worksheet. A new table named ‘Fast Go Game1’ is created.

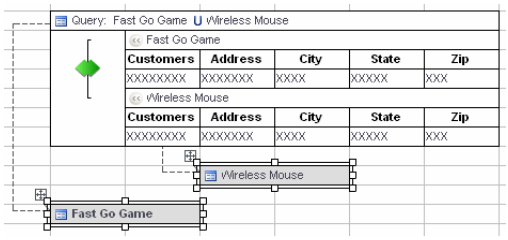
4. Drag the ‘Wireless Mouse’ node from the tree to an empty cell in the Worksheet. A new table named ‘Wireless Mouse1’ is created.



5. Ctrl-click the table title bars to select both tables.



6. Click on the ‘Concatenate Table’ button in the toolbar, and select ‘Union’. A new table, ‘Query1’, is created. Notice that the title row includes the names of the tables participating in the concatenation and the type of concatenation (a union symbol in this case).



7. Preview the table ‘Query1’ by right-clicking on its title row and selecting ‘Preview’ from the context menu. The information for all

customers who purchased more than 100 pieces of either ‘Wireless Mouse’ or ‘Fast Go Game’ is included in the table.

	A	B	C	D	E
1	Query				
2	CUSTOMERS	Address	City	State	Zip
3	Computer Tec	90 First Ave.	Los Angeles	CA	11223
4	Folk Mining	345 Sewgen I	Tucson	AZ	11234
5	Old World Insu	1230 Hoes La	New York	NY	08854
6	Interstate Sho	384 Broad St.	Wayne	NJ	18322
7	Northwest Ou	17 Main St.	Olympia	WA	93717

¢

4.1.3 Intersect

The intersection of two tables contains the unique set of *common* rows shared by the two tables, ordered lexicographically. Only a single instance of each shared row is retained in the intersection table; all duplicates are removed. For information on how to create an intersection table, see *Creating a Concatenated Table* or the example below.

Example: Intersection

Assume that you want a list of only those customers who have purchased more than 100 pieces of both ‘Wireless Mouse’ and ‘Fast Go Game’. This can be accomplished by taking the intersection of the ‘Wireless Mouse’ and ‘Fast Go Game’ queries.



1. Create a new Worksheet by clicking on the ‘New Worksheet’ button

2. Expand the ‘Data Source’ node, the ‘Orders’ node, and the ‘DWS’ node.

3. Drag the ‘Fast Go Game’ node from the tree to an empty cell in the Worksheet. A new table named ‘Fast Go Game1’ is created.

4. Drag the ‘Wireless Mouse’ node from the tree to an empty cell in the Worksheet. A new table named ‘Wireless Mouse1’ is created.

Fast Go Game1 (Primary)				
Customers	Address	City	State	Zip
XXXXXXXX	XXXXXXX	XXXX	XXXXX	XXX

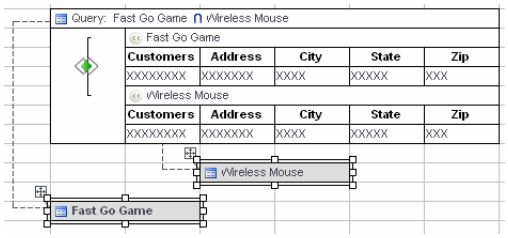
Wireless Mouse1				
Customers	Address	City	State	Zip
XXXXXXXX	XXXXXXX	XXXX	XXXXX	XXX

5. Now select both the tables by holding down the Ctrl key and clicking on each table’s title row.



6. Click on the ‘Concatenate Table’ button in the toolbar and select ‘Intersect’. A new table, ‘Query1’, is created. Notice that the title row includes the names of the tables participating in the

concatenation and the type of concatenation (an intersect symbol in this case).



7. Preview the table ‘Query1’ by right-clicking on its title row and selecting ‘Preview’ from the context menu. The information for only those customers who purchased more than 100 pieces of both ‘Wireless Mouse’ and ‘Fast Go Game’ is included in the table.

	A	B	C	D	E
1	Query1				
2	CUSTOMERS	Address	City	State	Zip
3	Folk Mining	345 Sewgen I	Tucson	AZ	11234
4	Old World Insi	1230 Hoes La	New York	NY	08854

¢

4.1.4 Minus

The difference (minus) of two tables is the set of distinct rows found in one table but not in the other. The first table selected is the table whose unique rows are retained in the result. For information on how to create a Minus table, see the *Creating a Concatenated Table* section or the example below.

Example: Minus

Assume you want a list of all the customers who purchased more than 100 pieces of ‘Fast Go Game’ but did not purchase more than 100 pieces of ‘Wireless Mouse’. Follow these steps to accomplish that result.

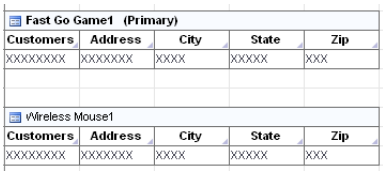


1. Create a new Worksheet by clicking on the ‘New Worksheet’ button.

2. Expand the ‘Data Source’ node, the ‘Orders’ node, and the ‘DWS’ node.

3. Drag the ‘Fast Go Game’ node from the tree to an empty cell in the Worksheet. A new table named ‘Fast Go Game1’ is created.

4. Drag the ‘Wireless Mouse’ node from the tree to an empty cell in the Worksheet. A new table named ‘Wireless Mouse1’ is created.

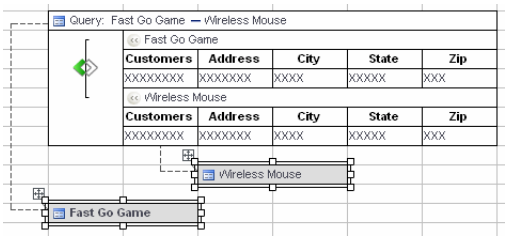


5. First select the ‘Fast Go Game1’ table by clicking the table’s title bar.

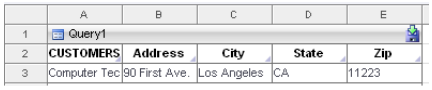
6. Now select the ‘Wireless Mouse1’ table as well by Ctrl-clicking the table’s title bar (It is important to select the ‘Wireless Mouse1’ table after selecting the ‘Fast Go Game1’ table.)



7. Click on the ‘Concatenate Table’ button, and select ‘Minus’. A new table, ‘Query1’, is created. Notice that the title row includes the names of the tables participating in the concatenation and the type of concatenation (a minus symbol in this case).



8. Preview the table ‘Query1’ by right-clicking on its title row and selecting ‘Preview’ from the context menu. The information for all customers who purchased more than 100 pieces of ‘Fast Go Game’ but did not purchase more than 100 pieces of ‘Wireless Mouse’ is included in the table.



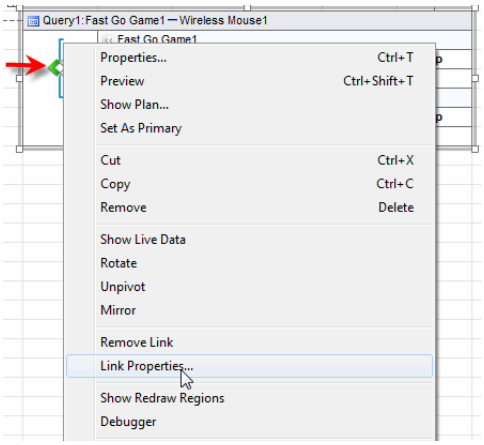
¢

4.1.5 Changing the Concatenation Type

To change the concatenation type of a Concatenated Table, follow these steps:

1. If the table is in Default view, switch the table to Composition view. (Press the ‘Change View’ button in the table title bar, and select ‘Composition’ from the menu. See *Editing a Composition Table* for more details.)

2. Select and then right-click the concatenation symbol or the “link” (the thin line) on the left side of the table, and select ‘Link Properties’ from the context menu. This opens the ‘Concatenation’ dialog box.



3. Select a new concatenation type from the menu in the ‘Concatenation’ dialog box.

4. If you set the concatenation type to ‘Union’, you can choose ‘Keep Duplicate Rows’ to prevent the removal of duplicate rows from the result.

5. (Optional) If the data block contains multiple concatenations, select ‘Apply to all concatenations’ to simultaneously update all of the links to the specified concatenation type. Leave this option disabled to modify only the selected link.

See Also

Creating a Concatenated Table, to create a new Concatenated Table.
Union, *Intersect*, and *Minus*, for information on the concatenation types.

4.2 Joining Tables

A *joined table* is a composition of two tables that preserves a specified relationship between the participating tables (*base tables*).

The joined table contains all the columns from both base tables, but retains only the rows satisfying a specified *join condition*. The join condition is a relation between two columns, one from each base table. These two columns are called the *join columns*. The joined table can be thought of as the intersection of the two tables based on the join columns.

There are no restrictions on the nature of the base tables for joins, and joined tables can themselves be used in subsequent join operations. There is no limit to the number of tables that can be successively joined together

in this way. Additionally, two tables can be joined together using multiple join conditions, that is, multiple pairs of join columns. This allows you to construct a joined table that embodies a very complex relation between the base tables.

The next sections explain how to create various types of joined tables.

See Also
What-If Analysis, for an illustration of using multiple join conditions.
Changing the Join Type, for information on converting between join types.
Editing a Composition Table, for Editable and Hierarchical modes of tables.

4.2.1 Inner Join

The most commonly-used join type is the *inner join*. The inner join of two tables can be described in the following way: "First pair *every* row of the first base table with *every* row of the second base table (Cartesian product), and then delete any row in the product where the values in the join columns do not satisfy the specified join condition." See the *Joining Tables* section for more general information about joins.

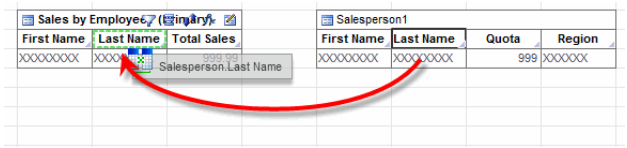
You can create an inner join either by dragging columns or by using the 'Join Table' button.

Creating a Inner Join by Dragging Columns

To create an inner join between two base tables by dragging columns, follow these steps:

1. Identify the join columns, one from each table. These are the columns to which the join condition is applied.

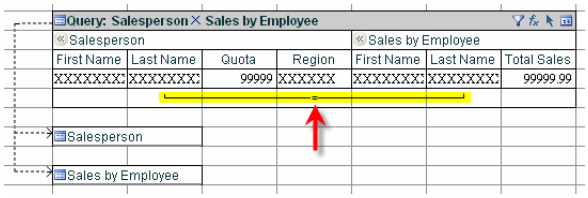
2. Drag the first join column header on top of the second join column header. The cursor should change to a join symbol, and a green line should appear in the second join column header.



3. When you see the join symbol and the green line, drop the column header.

   This creates a new table that represents the inner join of the two base tables. A graphical link is drawn between the two join columns, and the join type symbol is displayed. By default, the "equality" join condition is used, producing the so-called "equi-join." For

information on changing the join condition or join columns, see *Changing the Join Type*.



4. If desired, repeat the above steps to implement multiple join conditions.

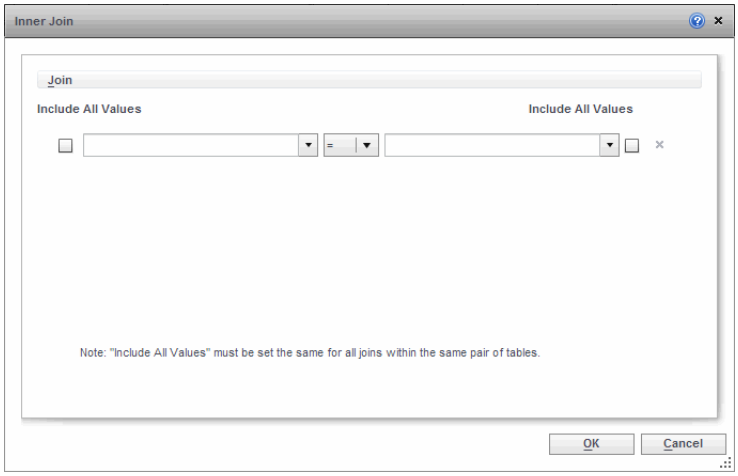
Creating an Inner Join with the 'Join Table' Button

To create an inner join between two base tables, follow these steps:

1. Ctrl-click to select the two table that will participate in the join.



2. Click the 'Join Table' button in the toolbar, and select 'Inner Join'. This opens the 'Inner Join' dialog box.



3. Use the menus to select one column from each table, and choose the join condition that the data must satisfy in order to appear in the result set. The '=' operator produces the "equi-join." The inequality operators produce their respective "non-equijoins."

4. (Optional) To convert an inner join to an outer join, select 'include all values' for the left or right table, or for both tables. These selections correspond, respectively, to the left outer join, the right outer join, and the full outer join.

Additional menus will appear as needed to specify the additional joins.

5. Select additional columns and join conditions from the remaining menus in the ‘Inner Join’ dialog box to create multiple join conditions.

The outer join is a generalization of the inner join: In addition to the matching rows preserved by the equi-join, the left outer join preserves all the rows in the left table, while the right outer join preserves all the rows in the right table. The full outer join preserves all the rows of both tables.

When you specify an outer join, the additional rows included from a table (beyond those selected by the equi-join) do not have matching rows in the other table. Therefore, outer-join tables generally exhibit empty cells corresponding to these unmatched rows.

For information on changing the join condition or join columns after creating the table, see *Changing the Join Type*.

Example: Creating an Inner Join

Suppose you have two queries, ‘Salesperson’ and ‘Sales by Employee’. The ‘Salesperson’ query returns the names of the sales people and their sales quotas. The ‘Sales by Employee’ query returns the names of the sales people and the total sales each one of them has made. You can very easily compare their total sales with the sales quota by joining the two tables, each of which is bound to one of the above mentioned queries.



1. Create a new Worksheet by clicking on the ‘New Worksheet’ button.

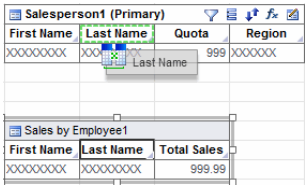
2. Expand the ‘Data Source’ node, then the ‘Orders’ node, and the ‘Order Model’ node.

3. Drag the ‘Salesperson’ entity on to one of the empty cells in the Data Worksheet. A new table, ‘Salesperson1’ is created.

4. Drag the ‘Sales by Employee’ query from the tree on to one of the empty cells in the Data Worksheet. A new table, ‘Sales by Employee1’ is created.

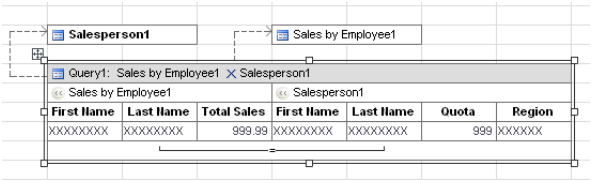


5. Now hold down the left mouse button and drag the ‘Last Name’ column header from the ‘Sales by Employee1’ table on top of the ‘Last Name’ column header of the ‘Salesperson1’ table. Notice that the cursor changes and a green line is drawn in the destination cell.



Notice that the connector line on the bottom row of the table shows the join relationship by connecting the join columns together.

6. Release the left mouse button to create the join. A new joined table, ‘Query1’, is created.



7. Preview the ‘Query1’ table by right-clicking on the title row and selecting ‘Preview’.

	A	B	C	D	E	F	G
1	Query1						
2	First Name	Last Name	Total Sales	First Name_1	Last Name_1	Quota	Region
3	Annie	Duke	4412983.00	Annie	Duke	250000.00	USA West
4	Eric	Heggenbart	5109322.00	Eric	Heggenbart	250000.00	USA East
5	Robert	Miller	3699185.00	Robert	Miller	150000.00	USA West
6	Sue	Marston	4435364.00	Sue	Marston	150000.00	USA East

¢

4.2.2 Outer Join

When you join two tables, the default join type created is an *equi-join*. This join type preserves only the rows of the two tables that have matching values on the join columns.

Sometimes, in addition to preserving the matching rows, you may also want to preserve *all* the rows of one or both base tables. A join which preserves all rows of one or both base tables is called an *outer join*.

See *Changing the Join Type* to change an inner join to an outer join.

See Also

Inner Join, for information about the default equi-join.

Changing the Join Type, to change an inner join to outer join or change the type of outer join.

4.2.3 Cross Join

Tables that do not share the same data model or query can be joined together with a *cross join*. A cross-joined table contains the cross-product (Cartesian product) of the data in the base tables. This means that every row in the first table is paired together with every row in the second table. The length of the new joined table is the product of the two base table lengths.

To cross-join two tables in a Data Worksheet, follow these steps:

1. Drag the first table alongside the second table.



2. Position the dragged table so that its border touches the border of the second table. The join icon should appear.

3. When the cross-join icon appears, release the dragged table.

The two tables are cross-joined to create a new table. Filtering conditions from both base tables are applied to the cross-joined table.

If the two base tables are *mergeable* (i.e., share the same data model or query), the steps above will produce a *merged* table rather than a cross-joined table. See *Merging Tables* for more information on this type of table manipulation.

Example: Cross Join

Say you have two queries, 'Salesperson' and 'Sales by Employee'. The 'Salesperson' query returns the names of the sales people and their sales quotas. Cross joining the tables corresponding to these two queries will create a joined table with a number of rows equal to the product of the rows of the two base tables.



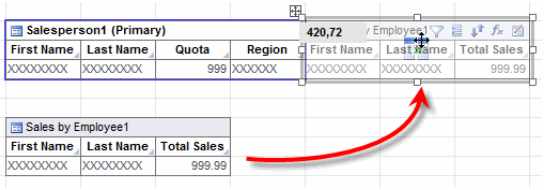
1. Create a new Worksheet by clicking on the 'New Worksheet' button.

2. Expand the 'Query node, then the 'Orders' node, and the 'Order Model' node.

3. Drag the 'Salesperson' entity on to one of the empty cells in the Data Worksheet. This creates a new table, 'Salesperson1'.

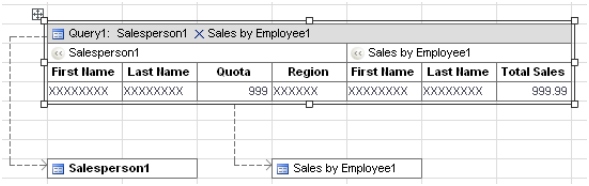
4. Drag the 'Sales by Employee' query from the tree on to one of the empty cells in the Data Worksheet. This creates a new table, 'Sales by Employee1'.

5. Now click on the title row of the 'Sales by Employee1' table, hold down the left mouse button and drag it close to the 'Salesperson1' table so that the two tables are laid side-by-side.





6. When the 'merge' icon appears, release the mouse to create the join. A new joined table, 'Query1', is created.



7. Preview the ‘Query1’ table by right-clicking on the title row and selecting ‘Preview’ from the context menu. Notice that the ‘Query1’ table contains the cross product of the data from the ‘Salesperson1’ and ‘Sales by Employee1’ tables.

	A	B	C	D	E	F	G
1	Query1						
2	First Name	Last Name	Quota	Region	First Name_	Last Name_	Total Sales
3	Sue	Marston	150000.00	USA East	Annie	Duke	4412983.00
4	Eric	Heggenbart	250000.00	USA East	Annie	Duke	4412983.00
5	Robert	Miller	150000.00	USA West	Annie	Duke	4412983.00
6	Annie	Duke	250000.00	USA West	Annie	Duke	4412983.00
7	Sue	Marston	150000.00	USA East	Eric	Heggenbart	5109322.00
8	Eric	Heggenbart	250000.00	USA East	Eric	Heggenbart	5109322.00
9	Robert	Miller	150000.00	USA West	Eric	Heggenbart	5109322.00
10	Annie	Duke	250000.00	USA West	Eric	Heggenbart	5109322.00
11	Sue	Marston	150000.00	USA East	Robert	Miller	3699185.00
12	Eric	Heggenbart	250000.00	USA East	Robert	Miller	3699185.00
13	Robert	Miller	150000.00	USA West	Robert	Miller	3699185.00
14	Annie	Duke	250000.00	USA West	Robert	Miller	3699185.00
15	Sue	Marston	150000.00	USA East	Sue	Marston	4435364.00
16	Eric	Heggenbart	250000.00	USA East	Sue	Marston	4435364.00
17	Robert	Miller	150000.00	USA West	Sue	Marston	4435364.00
18	Annie	Duke	250000.00	USA West	Sue	Marston	4435364.00

¢

4.2.4 Merge Join

The Merge Join is a special type of join that creates no relationship between the two base tables. The base tables are simply merged into a single table by collecting their columns together and placing corresponding rows side by side.

Note: A Merge-Joined table is not the same as a Merged table. See *Merging Tables* for information about merging tables that share the same query or model.

To create a merge join, follow these steps:

1. Select both tables by Ctrl-clicking on each table’s title row.



2. Click the ‘Join Table’ button in the Worksheet toolbar, and select ‘Merge Join’. A new merge-joined table is created.

In the merge-joined table, the columns of the shorter table are padded with empty rows to match the length of the longer table. The number of columns in the final merge-joined table is the sum of the columns in the two base tables.

Example: Merge Join

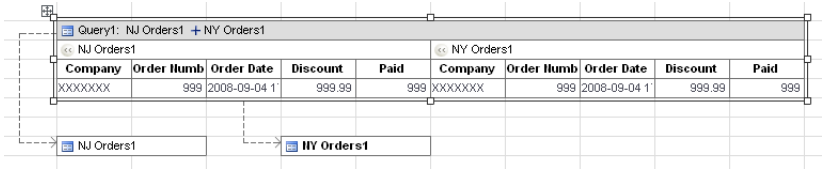
For example, say you want to see all of the orders from the states of NY and NJ side by side. You can accomplish this by using the merge join.



1. Create a new Worksheet.
2. Expand the ‘Data Source’ node, the ‘Orders’ node, and the ‘DWS’ node.
3. Drag the ‘NY Orders’ node from the tree to an empty cell in the Worksheet. A new table named ‘NY Orders1’ is created.
4. Drag the ‘NJ Orders’ node from the tree to an empty cell in the Worksheet. A new table named ‘NJ Orders1’ is created.
5. Now select both the tables by holding down the Ctrl key and clicking on each table’s title row.



6. Click on the ‘Join Table’ button on the top toolbar, and select ‘Merge Join’. A new table, ‘Query1’ is created.



Since the merge join does not create any relationship between the tables, there is no connector shown. This join type, therefore, cannot be changed.

7. Preview the ‘Query1’ table by right-clicking on the title row and selecting ‘Preview’. Notice that since the ‘NJ Orders1’ table has more rows than the ‘NY Orders1’ table, the number of rows of the ‘Query1’ table is equal to the number of rows of the ‘NJ Orders1’ table. Also notice that the ‘NY Orders1’ table has two empty rows appended to it.

	A	B	C	D	E	F	G	H	I	J
1	Query1									
2	Company	Order Numb	Order Date	Discount	Paid	Company_1	Order Numb	Order Date_	Discount_1	Paid_1
3	Eastern Data	12485	2011-11-06 0	0.10	1	Interstate Sho	12261	2010-09-04 0	0.25	1
4	Eastern Data	12668	2012-10-22 0	0.05	1	Interstate Sho	12455	2011-09-08 0	0.10	1
5	Old World Insu	12367	2011-03-28 0	0.05	1	Daycare One	12208	2010-05-22 0	0.25	1
6	Old World Insu	12453	2011-09-04 0	0.25	1	Daycare One	12551	2012-03-12 0	0.25	1
7	Old World Insu	12473	2011-10-15 0	0.15	1	Scheggie Ser	12697	2012-12-25 0	0.10	1
8	Old World Insu	12518	2012-01-09 0	0.25	1	Software Spe	12155	2010-01-22 0	0.20	1
9	Old World Insu	12559	2012-04-01 0	0.05	1	Software Spe	12416	2011-06-16 0	0.25	1
10	Old World Insu	12666	2012-10-18 0	0.10	1	Software Spe	12443	2011-08-13 0	0.25	1
11	Old World Insu	12676	2012-11-10 0	0.10	0	Software Spe	12653	2012-09-27 0	0.25	1
12	FISGA Corp	12386	2011-04-15 0	0.10	0	Rutgers Bank	12233	2010-07-08 0	0.10	1
13	FISGA Corp	12589	2012-05-23 0	0.25	1	Rutgers Bank	12425	2011-07-03 0	0.10	1
14	FISGA Corp	12660	2012-10-11 0	0.25	1	Rutgers Bank	12561	2012-04-01 0	0.10	1
15						Pharma Rese	12334	2011-01-11 0	0.15	1
16						Pharma Rese	12351	2011-02-14 0	0.20	1

¢

4.2.5 Changing the Join Type

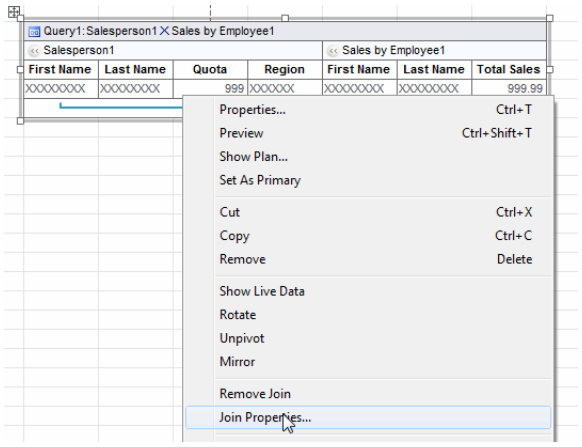
The join type can be changed for all joined tables except for Merge-joins and Cross-joins. (Merge joins and cross joins do not define a relationship between tables, so there is no relationship that can be changed.)

To change the join type of a joined table, follow these steps:

1. If the joined table is in Default view, switch the table to Composition view. (Press the 'Change View' button in the table title bar, and select 'Composition' from the menu. See *Editing a Composition Table* for more details.)

2. Open the 'Join Types' dialog box. You can do this in any one of the following ways:

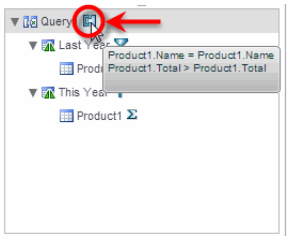
   a. Click the join connector link to select it. The selected link is highlighted in blue. Right-click on the highlighted join connector link and select 'Join Properties' from the context menu. This opens up the 'Join Types' dialog box for the selected join.



Additional menus will appear as needed to specify any additional joins.

   b. Right-click on the title bar of the Data Block and selected 'Join Properties'. This opens up the 'Join Types' dialog box for all joins.

   c. Press the 'Join' symbol next to the Data Block name in the Worksheet Explorer. (See *Navigating a Data Worksheet* for more information about the Worksheet Explorer.) This opens up the 'Join Types' dialog box for all joins.

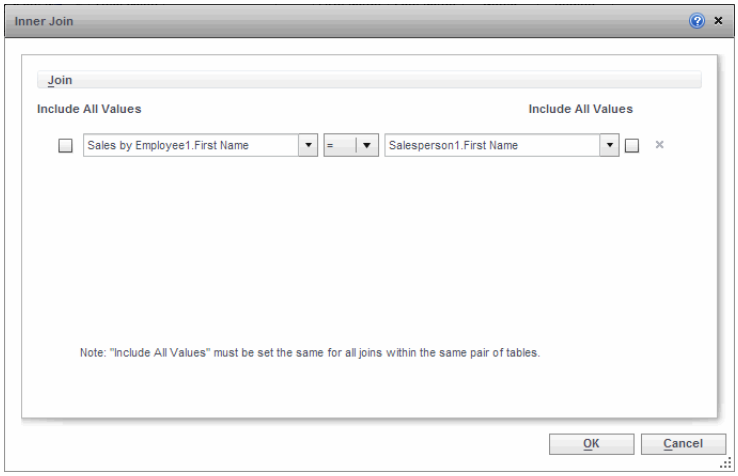


3. In the ‘Join Types’ dialog box, select the join columns from the drop-down menus for each table.

4. Select the join condition operator from the middle menu. The ‘=’ operator produces the “equi-join.” The inequality operators produce their respective “non-equijoins.”

5. (Optional) To convert an equi-join to an outer join, select ‘include all values’ for the top or bottom table, or for both tables.

   These selections correspond, respectively, to the left outer join, the right outer join, and the full outer join. In addition to the matching rows preserved by the equi-join, the left outer join preserves all the rows in the top table, while the right outer join preserves all the rows in the bottom table. The full outer join preserves all the rows of both tables.



When an outer join is specified, the additional rows included from a table (beyond those selected by the equi-join) do not have matching rows in the other table. Therefore, outer-join tables generally exhibit empty cells corresponding to the unmatched rows.

Example: Changing the Join Type

Follow the steps below to change the join type of a joined table from inner join to left outer join.



1. Create a new Worksheet by clicking on the ‘New Worksheet’ button.

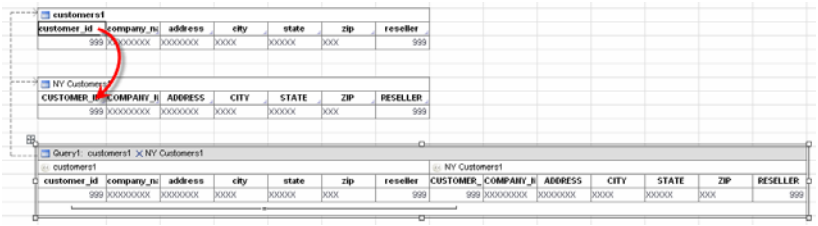
2. Expand the ‘Data Source’ node, and the ‘Orders’ node.

3. Drag the ‘customers’ query from the tree to an empty cell in the Worksheet. A new table named ‘customers1’ is created.

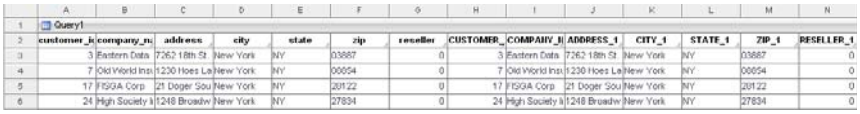
4. Expand the ‘DWS’ node on the ‘Orders’ tree.

5. Drag the ‘NY Customers’ query from the tree to an empty cell in the Worksheet. A new table named ‘NY Customers1’ is created.

6. Select the ‘customer_id’ column header in table ‘customers1’, hold down the left mouse button and drag it from the ‘customers1’ table onto the ‘CUSTOMER_ID’ column header of the ‘NY Customers1’ table. This creates a new (inner) joined table called ‘Query1’.



7. Right-click on the title row of the ‘Query1’ table and select ‘Preview’ from the context menu.

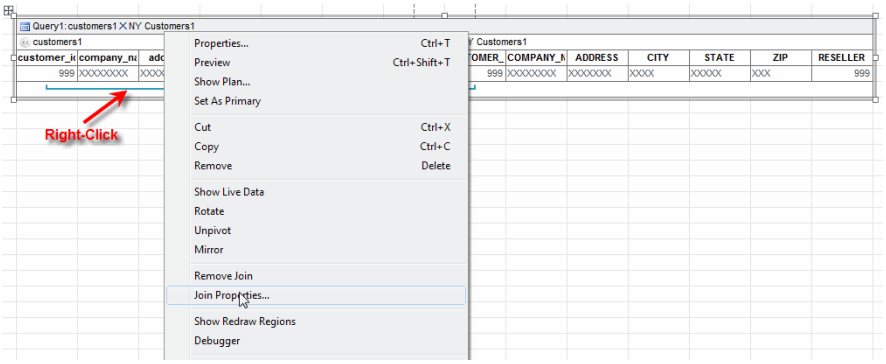


   Notice that only the customers from the state of NY are listed.

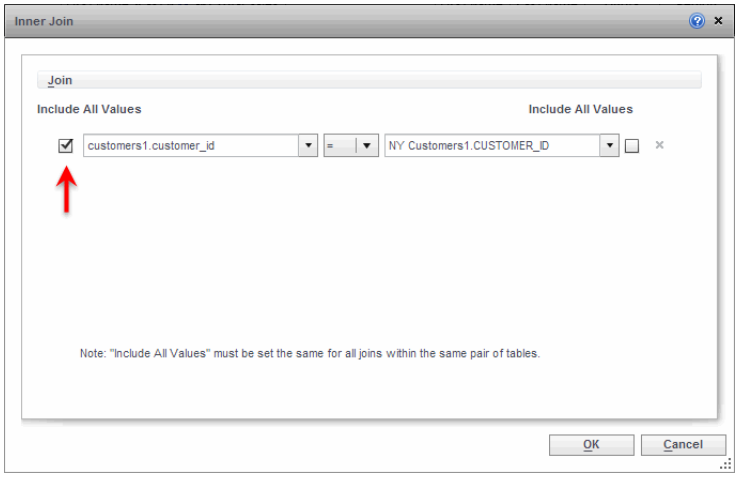
8. Close the preview.

9. In the Data Worksheet, click the join connector in table ‘Query1’ to select it.

10. Right-click the connector and select ‘Join Properties’ from the context menu. This opens the ‘Join Types’ dialog box.



11. Select the ‘include all values’ option for the ‘customers1’ table and press ‘OK’.



12. Preview the table again. Notice that the two tables are now joined using the left outer join.



¢

4.3 Merging Tables

Tables that are linked to the same query or data model are called *mergeable tables*. Two mergeable tables can be merged into a single table while preserving the original relationships in the data. This simply means that rows in the two tables that correspond to the same data record are merged together. The merge effectively provides just another view of the same query or data model, and does not entail any new joins.

To merge two mergeable tables in a Data Worksheet, follow these steps:

1. Drag the first table to the left or right side of the second table. Position the dragged table so that its border touches the border of the second table.



2. When the merge icon appears, release the dragged table. The dragged table will be merged into the second table.

When the tables are merged, the columns of the first (dragged) table are added to the right side of the second table, with corresponding rows of the two tables merged together. (The first table is left unaltered.) Filtering conditions that are defined on the second table are retained in the merged table, but filtering conditions defined on the first table are dropped. A merged table cannot be split.

If two tables are not mergeable, the above steps will produce a new *cross-joined* table rather than a merged table. See the *Cross Join* section for more information.

Note that a Merged table is *not the same* as Merge-Joined table. A Merge Join simply appends the columns of one table to another, with no relationship governing the row pairing. See the *Merge Join* section for more information.

Example:
Merging Tables

The following example illustrates the difference between merging tables and cross-joining tables. First, consider the case of two mergeable tables (tables linked to the same query or a data model):



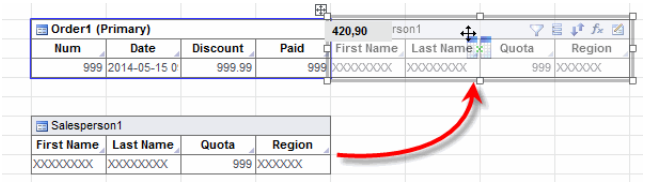
1. Create a new Worksheet by clicking on the ‘New Worksheet’ button.

2. Expand the ‘Data Source’ node, then the ‘Orders’ node, and the ‘Order Model’ node.

3. Drag the ‘Order’ entity on to one of the empty cells on the Worksheet. A new table, ‘Order1’ is created.

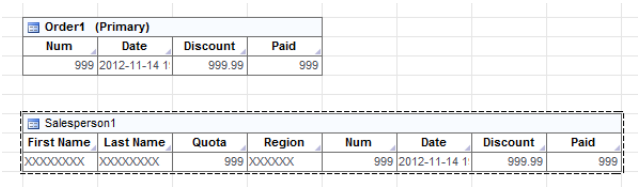
4. Drag the ‘Salesperson’ entity on to one of the empty cells in the Worksheet. A new table, ‘Salesperson1’ is created.

5. Now click on the title row of the ‘Salesperson1’ table, hold down the left mouse button and drag it near the ‘Order1’ table so that the two tables are laid side-by-side. Notice the merge icon which indicates that the two tables will be merged together.



Merged tables cannot be split.

6. Release the left mouse button to merge the two tables. Because the ‘Order1’ and ‘Salesperson1’ tables are bound to entities from the same data model, the ‘Order1’ table is modified to include the columns of the ‘Salesperson1’ table. No new tables are created.



7. Right-click on the title row of the ‘Salesperson1’ table and select ‘Preview’ to see the merged table.

	A	B	C	D	E	F	G	H
1	Salesperson1							
2	First Name	Last Name	Quota	Region	Num	Date	Discount	Paid
3	Sue	Marston	150000.00	USA East	12004	2009-03-18 0	0.00	1
4	Sue	Marston	150000.00	USA East	12013	2009-04-05 0	0.00	1
5	Sue	Marston	150000.00	USA East	12017	2009-04-12 0	0.00	1
6	Sue	Marston	150000.00	USA East	12024	2009-04-20 0	0.00	0
7	Sue	Marston	150000.00	USA East	12038	2009-05-15 0	0.00	1
8	Sue	Marston	150000.00	USA East	12041	2009-05-25 0	0.00	0
9	Sue	Marston	150000.00	USA East	12042	2009-05-27 0	0.00	1
10	Sue	Marston	150000.00	USA East	12047	2009-06-05 0	0.00	1
11	Sue	Marston	150000.00	USA East	12050	2009-06-08 0	0.00	1
12	Sue	Marston	150000.00	USA East	12051	2009-06-08 0	0.00	1
13	Sue	Marston	150000.00	USA East	12055	2009-06-19 0	0.00	1
14	Sue	Marston	150000.00	USA East	12058	2009-06-21 0	0.00	1
15	Sue	Marston	150000.00	USA East	12064	2009-06-29 0	0.10	1
16	Sue	Marston	150000.00	USA East	12071	2009-07-09 0	0.00	1
17	Sue	Marston	150000.00	USA East	12081	2009-08-01 0	0.00	1
18	Sue	Marston	150000.00	USA East	12090	2009-08-26 0	0.00	0
19	Sue	Marston	150000.00	USA East	12093	2009-08-30 0	0.00	1
20	Sue	Marston	150000.00	USA East	12095	2009-09-06 0	0.00	1
21	Sue	Marston	150000.00	USA East	12105	2009-09-28 0	0.00	1
22	Sue	Marston	150000.00	USA East	12108	2009-10-07 0	0.00	1
23	Sue	Marston	150000.00	USA East	12110	2009-10-12 0	0.00	0
24	Sue	Marston	150000.00	USA East	12112	2009-10-20 0	0.00	1
25	Sue	Marston	150000.00	USA East	12115	2009-10-30 0	0.00	1
26	Sue	Marston	150000.00	USA East	12116	2009-11-02 0	0.00	1

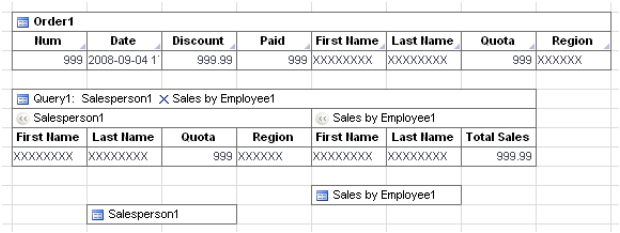
8. Close the ‘Preview’ window.

Next, consider the case of two non-mergeable tables (tables linked to different queries or a data models). See the *Cross Join* section for more information.

9. Drag the ‘Sales by Employee’ node on to one of the empty cells on the Worksheet. A new table, ‘Sales by Employee1’ is created.

10. Now click on the title row of the ‘Sales by Employee1’ table, hold down the left mouse button and drag it near the ‘Salesperson1’ table so that the two tables are laid side-by-side.

11. Release the left mouse button to cross-join the tables. Because the two tables do not share the same data model or a query, they are not merged together. Instead, a cross join is created between them, producing a new ‘Query1’ table.



Order1

Num	Date	Discount	Paid	First Name	Last Name	Quota	Region
999	2008-09-04 1	999.99	999	XXXXXXXX	XXXXXXXX	999	XXXXXX

Query1: Salesperson1 × Sales by Employee1

Salesperson1				Sales by Employee1		
First Name	Last Name	Quota	Region	First Name	Last Name	Total Sales
XXXXXXXX	XXXXXXXX	999	XXXXXX	XXXXXXXX	XXXXXXXX	999.99

Sales by Employee1

Salesperson1

12. Preview the ‘Query’ table.

	A	B	C	D	E	F	G
1	Query1						
2	First Name	Last Name	Quota	Region	First Name_	Last Name_'	Total Sales
3	Sue	Marston	150000.00	USA East	Annie	Duke	4412983.00
4	Eric	Heggenbart	250000.00	USA East	Annie	Duke	4412983.00
5	Robert	Miller	150000.00	USA West	Annie	Duke	4412983.00
6	Annie	Duke	250000.00	USA West	Annie	Duke	4412983.00
7	Sue	Marston	150000.00	USA East	Eric	Heggenbart	5109322.00
8	Eric	Heggenbart	250000.00	USA East	Eric	Heggenbart	5109322.00
9	Robert	Miller	150000.00	USA West	Eric	Heggenbart	5109322.00
10	Annie	Duke	250000.00	USA West	Eric	Heggenbart	5109322.00
11	Sue	Marston	150000.00	USA East	Robert	Miller	3699185.00
12	Eric	Heggenbart	250000.00	USA East	Robert	Miller	3699185.00
13	Robert	Miller	150000.00	USA West	Robert	Miller	3699185.00
14	Annie	Duke	250000.00	USA West	Robert	Miller	3699185.00
15	Sue	Marston	150000.00	USA East	Sue	Marston	4435364.00
16	Eric	Heggenbart	250000.00	USA East	Sue	Marston	4435364.00
17	Robert	Miller	150000.00	USA West	Sue	Marston	4435364.00
18	Annie	Duke	250000.00	USA West	Sue	Marston	4435364.00

¢

5 Manipulating Tabular Data

This section discusses various techniques for manipulating the data within a table.

5.1 Sorting

Any column in a table can be sorted in ascending or descending order. You can sort the table data based on a single column, or specify nested sorting columns. If a column in the table is already sorted, sorting on a new column clears the previous sorting.

5.1.1 Sorting a Single Column

To sort a single column, do one of the following:

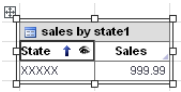
- Click the 'Sort' button in the column header. This sorts the column in ascending order. Click the button again to sort the column in descending order. Click a third time to return the column to its original order.
- Press the 'Sort Column(s)' button in the table title bar. This opens the 'Sort Column' dialog box, where you can specify the sort order for the selected column. See *Sorting Multiple Columns* for more information on using the dialog box.

When a column is sorted, its sort order is indicated by the small arrow next to the column name in the column header cell.

Example: Sorting a Single Column

Assume you have a table displaying the sales information for all states and you would like to sort the data based on the ascending order of the states. Follow these steps:

1. Create a new Worksheet.
2. Expand the 'Data Source' node, and the 'Orders' node.
3. Drag the 'sales by state' query on to an empty cell in the Worksheet. This creates a new table named 'sales by state1'.
4. Select the 'State' header cell.
5. Click on the 'Sort' button, located next to the column name in the column header cell. Notice the presence of an up arrow next to the 'state' column name in the table.

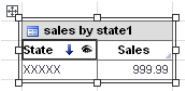


6. Preview the table.

	A	B	C
1	sales by state1		
2	State	Sales	
3	AZ	456333.85	
4	CA	1665880.05	
5	CO	838303.20	
6	CT	161689.50	
7	FL	377534.80	
8	IL	558702.80	
9	MA	2126975.20	
10	MD	1399917.10	
11	NJ	3219337.20	
12	NV	395348.70	
13	NY	2441696.65	
14	PA	420043.30	
15	TX	2465336.95	
16	WA	923016.25	

7. Close the ‘Preview’ window, and select the ‘State’ column header again.

8. Click on the ‘Sort’ button. Notice that the ‘State’ column will now be sorted in descending order. This is indicated by the down arrow present next to the column name.



9. Preview the table.

	A	B	C
1	sales by state1		
2	State	Sales	
3	WA	923016.25	
4	TX	2465336.95	
5	PA	420043.30	
6	NY	2441696.65	
7	NV	395348.70	
8	NJ	3219337.20	
9	MD	1399917.10	
10	MA	2126975.20	
11	IL	558702.80	
12	FL	377534.80	
13	CT	161689.50	
14	CO	838303.20	
15	CA	1665880.05	
16	AZ	456333.85	

Clicking on the ‘Sort’ button of the ‘State’ column a further time will result in the data being displayed in its original order.

10. Now select the ‘Sales’ column header cell.

11. Click on the ‘Sort’ button. Notice that the ‘Sales’ column is now sorted in the ascending order and the ‘State’ column is no longer sorted. The ‘State’ column is set to display the data in its original order.



12. Preview the table.

	A	B	C
1	sales by state1		
2	State	Sales	
3	CT	161689.50	
4	FL	377534.80	
5	NV	395348.70	
6	PA	420043.30	
7	AZ	456333.85	
8	IL	558702.80	
9	CO	838303.20	
10	WA	923016.25	
11	MD	1399917.10	
12	CA	1665880.05	
13	MA	2126975.20	
14	NY	2441696.65	
15	TX	2465336.95	
16	NJ	3219337.20	

¢

5.1.2 Sorting Multiple Columns

To sort multiple columns in a hierarchical fashion, follow these steps:



1. Press the ‘Sort Column(s)’ button in the table title bar. This opens the ‘Sort Column’ dialog box.

2. From the ‘Sort on’ drop-down menus, select the table columns that you wish to sort.

   The hierarchical sort order of a column is determined by the vertical position of the menu. The column specified in the top menu is sorted first, the column specified in the second-to-top menu is sorted second, and so on.

3. For each column, select the ‘Ascending’ or ‘Descending’ button to determine the sorting direction.

When a column is sorted, its sort order is indicated by the small arrow next to the column name in the column header cell. To sort just a single column, see *Sorting a Single Column*.

Example: Sorting Multiple Columns

Say you have a table listing the information of all of our customers and you would like to sort the data based on the state first, and then the city. To sort both the ‘state’ and the ‘city’ columns of the table, follow these steps:

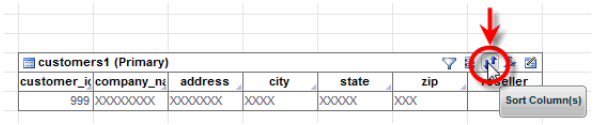


1. Create a new Worksheet.

2. Expand the ‘Data Source’ node, and the ‘Orders’ node.

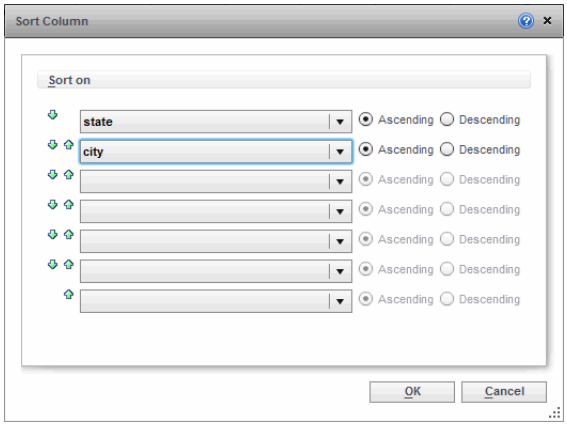
3. Drag the ‘customers’ query on to an empty cell in the Worksheet. This creates a new table named ‘customers1’.

4. Press the ‘Sort column(s)’ button in the table title bar. This opens the ‘Sort Column’ dialog box.



5. Select ‘state’ from the first drop-down list, and ensure that the ‘Ascending’ radio button is selected.

6. Select ‘city’ from the second drop-down list, and ensure that the ‘Ascending’ radio button is selected.



7. Click on the ‘OK’ button. Notice that the up arrows next to the column names ‘state’ and ‘city’ indicate that both of those columns are sorted in ascending order.



8. Preview the table. Notice that the data is sorted by the ‘state’ and the ‘city’ columns.

	A	B	C	D	E	F	G
1	customers1						
2	customer_id	company_na	address	city	state	zip	reseller
3	20	Folk Mining	345 Sewgen I	Tucson	AZ	11234	0
4	16	Whittaker & C	44 Islander Le	Albany	CA	83923	0
5	21	West Coast C	460 Newcomi	Los Angeles	CA	73646	0
6	13	Computer Tec	90 First Ave.	Los Angeles	CA	11223	1
7	9	Direct Sales	4500 River Rc	San Francisc	CA	08854	1
8	4	Ernst Handel	88 Rio Grand	Boulder	CO	38923	1
9	8	Quick Stop	71 Mulberry S	Denver	CO	98839	0
10	6	Loctite Corp.	5 Baker's Lan	Hartford	CT	39283	0
11	12	Fast Transpor	77 Druguz Blv	Orlando	FL	12343	0
12	10	Specialty Rete	88 Specialty D	Chicago	IL	88938	1
13	23	World Softwa	563 Uptown D	Boston	MA	28275	1
14	18	Ubermeyer	8300 Clevelar	Boston	MA	29183	0
15	34	XO Resellers	77 Huntington	Cambridge	MA	02938	1
16	30	AMG Logistics	283 Handley F	Springfield	MA	01923	0

¢

5.2 Filtering

You can filter the data in a table by applying conditions to any of the table columns. For example, you may want to display only products that have a 'product price' greater than $100, or that have an 'order date' within the past year. When you specify conditions, the table will only display the data records that satisfy those conditions.

See Also

Filter Conditions, in *Advanced Dashboard Design*, for information on setting a Viewsheet condition.

Using a Subquery in a Condition, for generating condition criteria at runtime.

Concatenating Tables, for filtering rows of a table based on set operations.

Joining Tables, for filtering table rows by joining two tables.

What-If Analysis, for implementing an adjustable filtering condition.

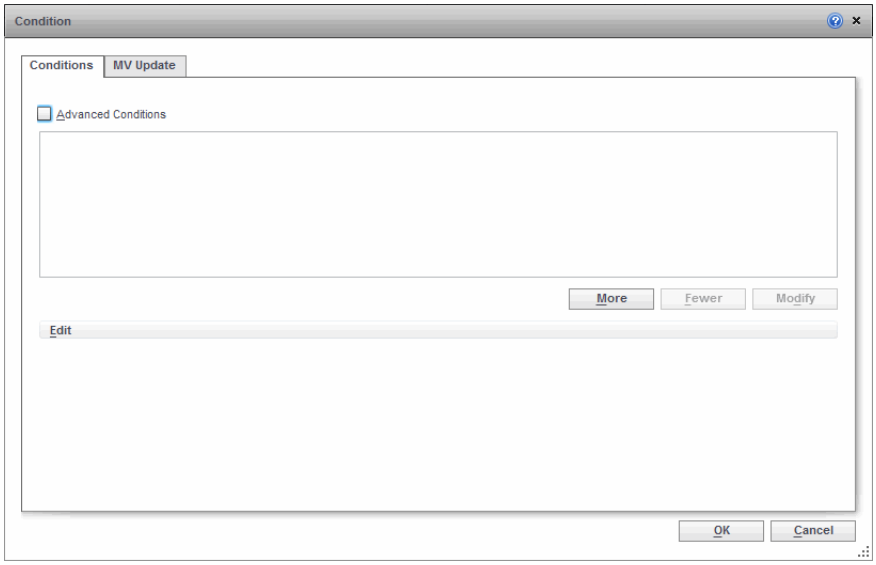
5.2.1 Defining Simple Conditions

Simple conditions are conditions defined on the pre-aggregate data, that is, before any grouping or summarization. Simple conditions are always joined together using the 'AND' operator. To gain greater flexibility, you can use advanced conditions; see *Defining Advanced Conditions*.

Note: Filtering conditions are only applied to data that is retrieved from the database. The amount of data (number of rows) retrieved is determined by the 'Max Rows' setting and the global 'Design mode sample data size' setting. See *Previewing a Table* and *Showing Live Data in a Table* for more information.

To define a simple condition, follow these steps:

1. Click the 'Condition' button on the title bar of a Data Table. This opens the 'Condition' dialog box.



2. Click the ‘More’ button to specify a new condition.

3. Use the menus in the ‘Edit’ pane to define the condition. See *Filtering Operators* for details on the options.

4. Click ‘OK’ to save the condition and close the dialog box. Click ‘Modify’ to update the condition without closing the dialog box.

5. To delete a condition, select the condition in the list and click the ‘Fewer’ button.

If the condition you specify references data model attributes that do not already exist in the table, these attributes are automatically added to the table as invisible columns.

Example: Defining Simple Conditions

Say you have a table listing the total sales of products grouped by their categories, and you wish to view only those categories that have yielded total sales of at least $1,000,000.

1. Create a new Worksheet.

2. Expand the ‘Data Source’ node, and the ‘Orders’ node.

3. Drag the ‘Sales by Category’ query on to an empty cell in the Worksheet. This creates a new table named ‘Sales by Category1’.

4. Click on the ‘Condition’ button located in the upper right corner of the table’s title bar. This will open the ‘Conditions’ dialog.

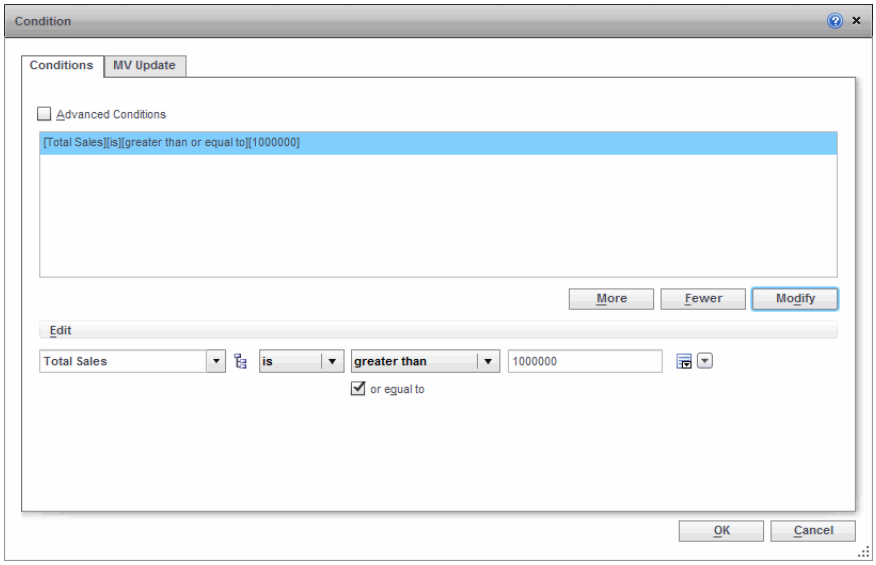
5. Click on the ‘More’ button to create a new condition.

6. Select ‘Total Sales’ from the drop-down list on the left.

7. Select ‘greater than’ from the operator drop-down list and check the ‘or equal to’ check box to add the ‘greater than or equal to’ condition.

A Variable or another field value can be used in the condition by selecting ‘variable’ or ‘field’ and specifying either the Variable or the field name. See *Creating a Variable* for more information.

8. Specify “1000000” for the value.



9. Click on the ‘OK’ button.

10. Right-click the table title bar, and select ‘Preview’ from the context menu.

	A	B	C	D	E
1	Sales by Category1				
2	Category	Qty Sold	Total Discou	Total Sales	
3	Business	4499	99787.50	8093500.00	
4	Hardware	8518	36712.00	4972310.00	
5	Office Tools	5057	17597.85	1126838.00	
6	Personal	6520	18331.85	1177798.00	

¢

5.2.2 Defining Advanced Conditions

To specify conditions on both pre-aggregate and post-aggregate data, as well as group rankings, use the ‘Advanced Conditions’ dialog box. Advanced conditions can additionally be joined using both ‘AND’ and ‘OR’ operations. If you only need pre-aggregate conditions and ‘AND’ operations, you can use simple conditions instead. See *Defining Simple Conditions* for more information.

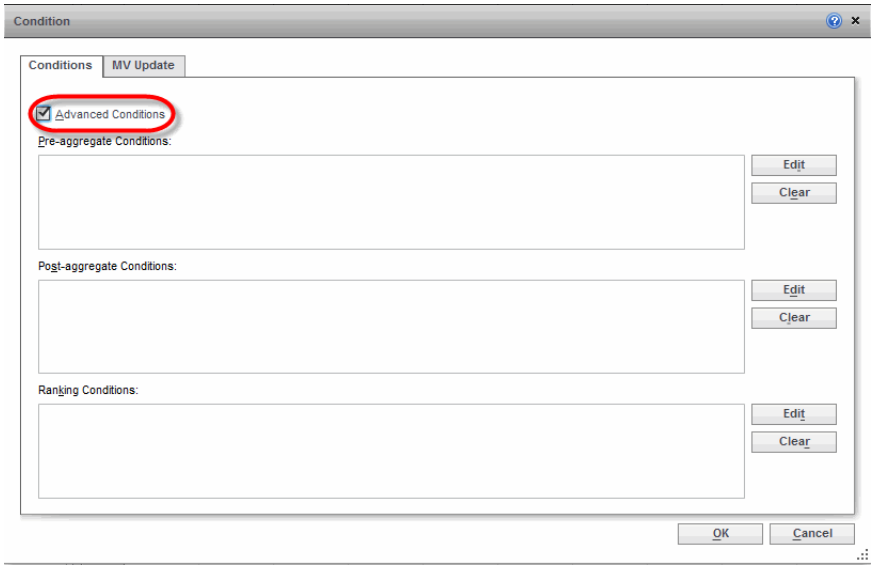
To define advanced conditions, follow these steps:



1. Click the ‘Condition’ button on the title bar of a Data Table. This opens the ‘Condition’ dialog box.

2. Select the ‘Advanced Conditions’ check box.

   The ‘Advanced Conditions’ check box can be used to toggle between simple and advanced conditions. However, when switching from the ‘Advanced conditions’ mode to the ‘Simple Conditions’ mode, if an advanced condition cannot be mapped to a simple condition, it will be permanently removed from the list of conditions.



3. Define a condition on pre-aggregate data, post-aggregate data, or ranking by clicking on the corresponding ‘Edit’ button. See *Pre-aggregate, Post-aggregate, and Ranking Conditions* below.

If the condition you specify references attributes that do not already exist in the table, these attributes are automatically added to the table as invisible columns.

Pre-aggregate, Post-aggregate, and Ranking Conditions

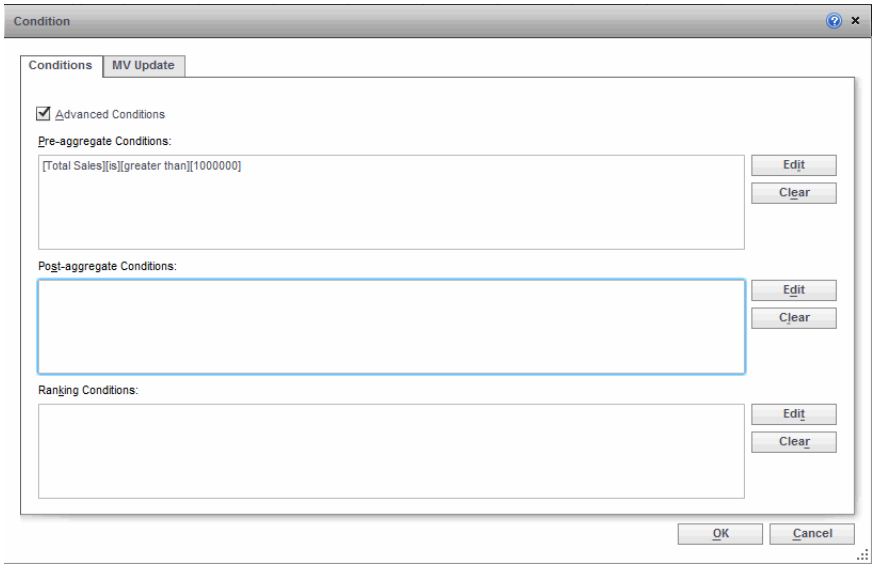
Pre-aggregate conditions are applied to the detail rows of the table, whereas post-aggregate conditions are applied to the data in the summary rows of the table. (Pre-aggregate conditions are equivalent to conditions defined in the ‘Where’ clause of the SQL statement, and post-aggregate conditions are equivalent to conditions specified in the ‘Having’ clause of the SQL statement.) Ranking conditions specify the Top N or Bottom N number of rows that should be presented in the table. See *Filtering Operators* for information about the filtering options.

Example: Defining Advanced Conditions

Say that you want to view the sales information of only the top three product categories in the table that you created in the previous section, *Defining Simple Conditions*.

1. Press the ‘Condition’ button located in the upper right corner of the table’s title bar. This opens the ‘Conditions’ dialog box.

2. Click on the ‘Advanced Conditions’ check box. Notice that the condition added to the ‘Total Sales’ field is included in the ‘Pre-aggregate’ conditions list.

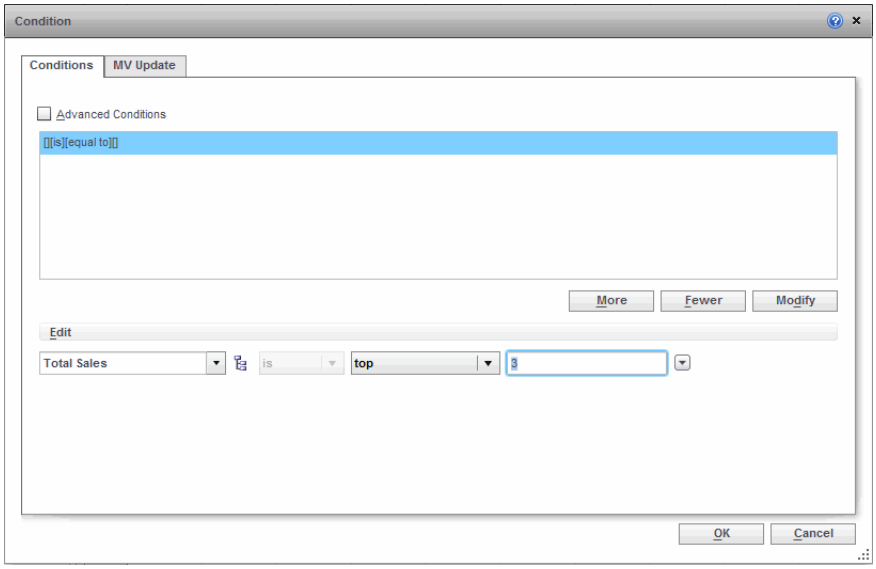


3. Click on the ‘Edit’ button for the ‘Ranking Conditions’ and click on the ‘More’ button.

4. From the drop-down list, select ‘Total Sales’.

5. Ensure that the ranking operation is set to ‘top’ and specify “3” as the value.

A Variable value can be used as the ranking condition value by selecting ‘variable’ from the drop-down list to the right of the value text box and specifying the parameter name in the text box. See *Creating a Variable* for more information.



6. Click on the ‘OK’ button to go back to the ‘Conditions’ dialog.

7. Click on the ‘OK’ button to apply all of the conditions to the table.

8. Preview the table. Data for only the top three categories, based on the ‘Total Sales’ value, is listed.

	A	B	C	D	E
1	Sales by Category1				
2	Category	Qty Sold	Total Discou	Total Sales	
3	Business	4499	99787.50	8093500.00	
4	Hardware	8518	36712.00	4972310.00	
5	Personal	6520	18331.85	1177798.00	

¢

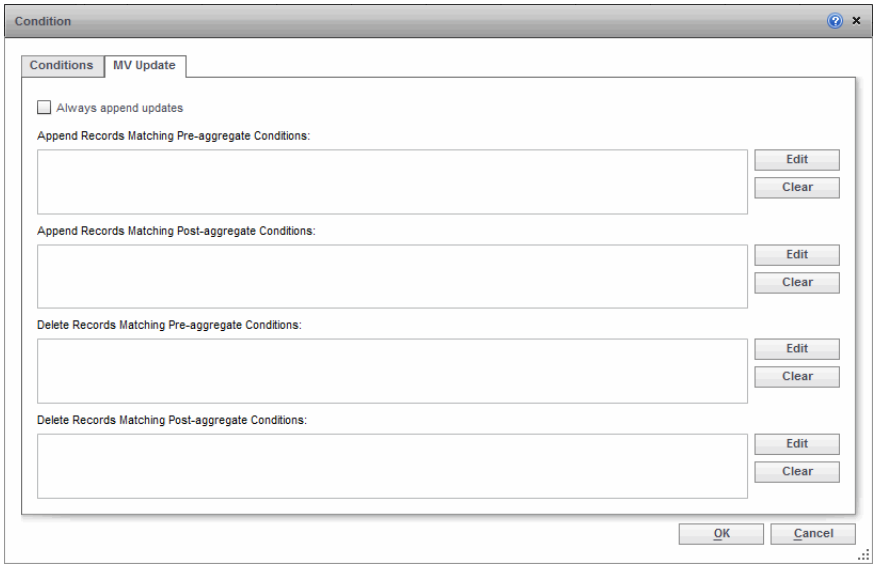
5.2.3 Defining a Materialized View Update Condition

You can create conditions for a Data Block which determine how and when the Data Block’s materialized view is updated. (See *Incrementally Updating a Materialized View* in *Advanced Server Administration* for background on incremental updates.)

To specify the materialized view update conditions for a Data Block, follow the steps below:

1. Press the ‘Condition’ button located in the upper right corner of the Data Block title bar. This opens the ‘Conditions’ dialog box.

2. Select the **MV Update** tab.



3. Press the ‘Edit’ button corresponding to the type of update condition you wish to create. The available options are as follows:

   a. **Append Records Matching Pre-aggregate Condition**: Create a condition which selects records to add to the existing materialized view based on the pre-aggregate values of data (prior to grouping and summarization).

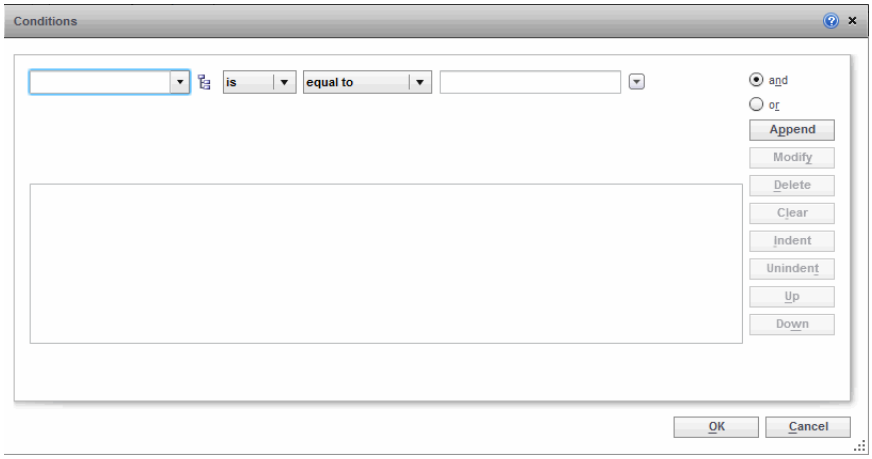
   b. **Append Records Matching Post-aggregate Condition**: Create a condition which selects records to add to the existing materialized view based on the post-aggregate values of data (subsequent to grouping and summarization).

   c. **Delete Records Matching Pre-aggregate Condition**: Create a condition which selects records to delete from the existing materialized view based on the pre-aggregate values of data (prior to grouping and summarization).

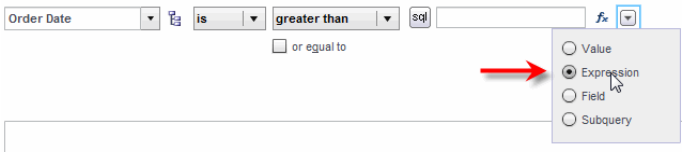
   d. **Delete Records Matching Post-aggregate Condition**: Create a condition which selects records to delete from the existing materialized view based on the post-aggregate values of data (subsequent to grouping and summarization).

   When you press the desired ‘Edit’ button, this opens the ‘Conditions’ dialog box.

4. Use the menus to enter the desired condition. (See *Defining Simple Conditions* for general information on specifying conditions.)



If you select the 'Expression' option for the right side of the condition, follow the additional steps below:



The 'sql/js' option is not available for deletions. Deletion expressions must use JavaScript syntax.

a. For append operations, press the 'sql/js' button to select the desired expression language, SQL or JavaScript.

b. Press the 'Edit Expression' button (*fx*). This opens the Formula Editor.

c. Enter the desired expression. You can use any available functions and operators, in addition to the special parameters `MV.LastUpdatedTime`, `MV.MaxValue`, and `MV.MinValue`. See *Special Parameters for Materialized View Update Conditions* for more details.

   Note: To prevent a delete condition from being applied on the initial materialized view generation, add the following special tag into the expression script: `//@incrementalOnly`.

d. Press 'OK' to close the Formula Editor.

5. Press 'OK' to close the 'Condition' dialog box.

The specified condition(s) will be evaluated when any materialized view based on this Data Block is regenerated. The materialized view will be updated in accordance with the conditions specified. In general, incrementally updating a materialized view in this way is significantly faster than regenerating the materialized view in its entirety.

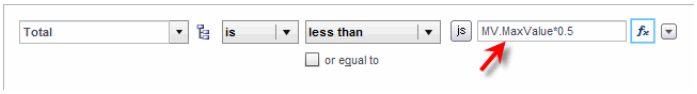
Special Parameters for Materialized View Update Conditions

The following special parameters are available for expressions in the materialized view update condition.

- `MV.LastUpdatedTime`: This represents the time at which the materialized view for the Data Block was last updated. You can use this value in an condition expression to incrementally update the materialized view with records that have posted to the database since the last materialized view update. For example, the condition below updates the current materialized view with records having a "transaction date" subsequent to the previous materialization.

  ```
  [transaction_date][is][greater than][MV.LastUpdatedTime]
  ```

- `MV.MaxValue`: This represents the maximum value of the field specified in the left-side menu of the materialized view condition. For example, in the condition below, `MV.MaxValue` represents the maximum value of the 'Total' field.



- `MV.MinValue`: This represents the minimum value in the field specified in the left-side menu of the materialized view condition.

5.2.4 Filtering Operators

The table below lists the operators available for use in filtering conditions.

OPERATOR	DESCRIPTION
equal to	True when the column value identically matches the specified criterion. (String matching is case-insensitive.)
in range	For date fields, true when the column value (date) falls within the specified date range criterion. See Appendix C.1, *Built-in Date Ranges* for a list of available date ranges.
one of	True when the column value is a member of the set of criteria values. (String matching is case-sensitive.)
less than	True when the column value is strictly less than the criterion value. Select the 'or equal to' check box to obtain a "less than or equal to" criterion. (Strings are compared based on their alphabetical ordering.)
greater than	True when the column value is strictly greater than the criterion value. Select the 'or equal to' check box to obtain a "greater than or equal to" criterion. (Strings are compared based on their alphabetical ordering.)

OPERATOR	DESCRIPTION
between	True when the column value is within the interval specified by its upper and lower boundaries. The boundary values are included in the interval. (Strings are compared based on their alphabetical ordering.) For date fields, consider using the 'in range' operator in place of specifying fixed dates.
null	True when the column value is 'null'. (Nulls appear as empty cells in a table.)
top	For grouped and aggregated data, true when the aggregated column value is among the top N aggregated values, according to the grouping specified by the secondary menu.
bottom	For grouped and aggregated data, true when the aggregated column value is among the bottom N aggregated values, according to the grouping specified by the secondary menu.
starting with	For numeric and string data, true when the column value begins with (i.e., the left-most characters match) the specified numerals or characters. String matching is case-insensitive.
contains	For numeric and string data, true when the column value contains the exact sequence of specified numerals or characters. String matching is case-insensitive.

5.2.5 Using a Parameter in a Condition

You can use a *variable* or *parameter* as the test value in a filter condition. By default, the user of the Worksheet (including users of linked Viewsheets and reports) will be prompted at runtime to enter the value of the parameter.

The value of the variable can also be supplied from the following sources:

- Viewsheet input component. (See *Passing Inputs to an Embedded Table or Variable* in *Advanced Dashboard Design*.)
- User session data. (See *SRPrincipal Properties* in *Advanced Server Integration*.)

To use a variable or parameter in a filter condition, follow these steps:

1. Open the Worksheet containing the table you want to filter.
2. (Optional) To reference a a *variable asset* in the condition, first create the Variable asset, or drag an existing Variable from the asset tree to an empty cell in the Worksheet. (See *Defining a Variable* for more information.)
3.  Click the 'Condition' button on the title bar of the table you wish to filter. This opens the 'Condition' dialog box.
4. Open the edit field in either the 'Simple Condition' mode (click the 'More' button) or 'Advanced Condition' mode (click the pre-aggregate or post-aggregate 'Edit' button).

5. To use a variable in the condition, select ‘Variable’ from the rightmost popup menu. This opens a blank Combo Box on the right side of the condition.

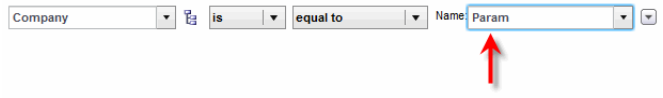


In the Combo Box, do one of the following:

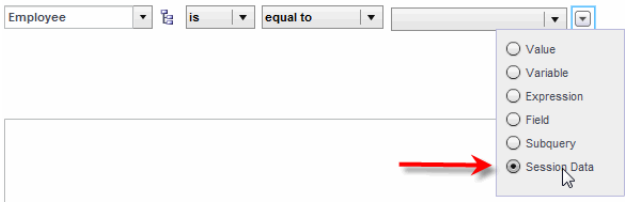
a. Select an existing Worksheet Variable from the menu.

b. Type the name of a new parameter.

c. Type the name of a Viewsheet input component.

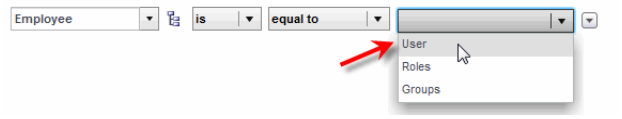


6. To use session data in the condition, select ‘Session Data’ from the rightmost popup menu. This opens a menu on the right side of the condition.



Note that the adjacent menu automatically chooses the ‘equal to’ or ‘one of’ operator to match your selection.

In the menu, choose one of ‘User’, ‘Roles’, or ‘Groups’. These parameters return information about the user who is currently accessing the Worksheet; respectively, the user name, the array of roles to which the user belongs, and the array of groups to which the user belongs.



7. Define the rest of the condition as described earlier.

The default prompting for Variables and parameters is the same. However, Variables provide additional control over the interface (radio button, menu, etc.) and the set of choices presented to the user.

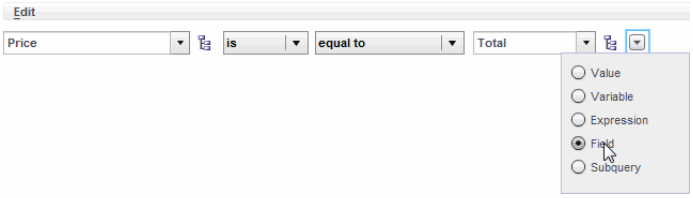
See Also
Filtering, for details on specifying filtering conditions.
Defining a Variable, for more information on defining a Variable.

5.2.6 Using a Field in a Condition

You can use table values as the test values in a filter condition. That is, you can define a filter condition that compares two values in the same row of the table.

To use a table field in a filter condition, follow these steps:

1. Open the Worksheet containing the table you want to filter.
2. Click the ‘Condition’ button on the title bar of the table you wish to filter. This opens the ‘Condition’ dialog box.
3. Open the edit field in either the ‘Simple Condition’ mode (click the ‘More’ button) or ‘Advanced Condition’ mode (click the pre-aggregate or post-aggregate ‘Edit’ button).
4. Make the desired selections in the left three menus.
5. Select ‘Field’ from the right-most menu. This enables a Combo Box on the right side of the condition.



6. In the Combo Box, select the table column containing the values to test.

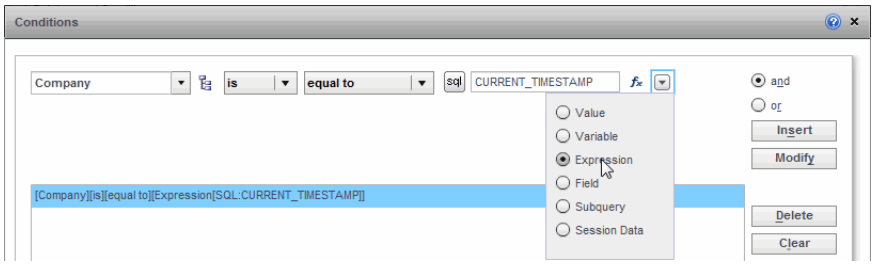
The condition compares corresponding values in the selected columns (left-most menu and right-most menu), and preserves only rows where the condition is met.

See Also
Filtering, for details on specifying filtering conditions.

5.2.7 Using an Expression in a Condition

The test values of a filtering condition can be a SQL or a JavaScript expression. To filter a field in a data block based on an expression, follow these steps:

1. Open the Worksheet containing the data block you want to filter.
2. Click the 'Condition' button on the title bar of the data block you wish to filter. This opens the 'Condition' dialog box.
3. Select the field and the desired logical operator.
4. Click on the arrow button to choose 'Expression' from the menu.



5. Click the 'sql' or 'js' button at the left of the expression text field to choose the expression type.

Note: Use SQL expressions if possible. See *SQL vs. JavaScript in Expressions*.

6. Click on the 'Fx' button to launch the expression edit dialog and enter your expression.

For example, to retrieve orders placed before today, you can use one of the following conditions:

```
// SQL:
[Order Date] [is] [less than] [CURRENT_TIMESTAMP]

// JavaScript:
[Order Date] [is] [less than] [CALC.today()]
```

SQL vs. JavaScript in Expressions

Expressions that use SQL syntax are processed by the database, while expressions that use JavaScript syntax are post-processed by the Worksheet. Because post-processing is CPU-intensive for the query engine, and may additionally cause retrieval of a large amounts of raw data, it is preferable to use SQL syntax when possible. Note, however, that SQL functions are database-specific and vary from one database to another.

5.2.8 Using a Subquery in a Condition

When you add a filter to a table, you can choose to use a *subquery* to return the test conditions. This allows you to design a filter with conditions that are set at runtime based on current data.

For example, consider a table with a column named 'CompanyName'. You want this table to display only companies in NY state, but the table has no 'State' column on which you can define the filter. If you knew *in advance* which companies reside in NY, you could enter the condition explicitly; for example,

```
[Company][is][one of][explicit list of companies in NY]
```

However, if the set of companies in NY is not fixed, and can therefore change whenever the underlying database is updated, any fixed list that you enter in the condition will become outdated. In order to filter the table correctly, you need to generate a list of "companies in NY" that is always true to the current data.

You can do this by using a *subquery* to provide the current list of "companies in NY." The condition you place on the table will then have the form

```
[Company][is][one of][results of subquery]
```

The list of "companies in NY" generated by the subquery will always be up-to-date, because the subquery itself executes at runtime.

To enter a subquery in a filter condition, follow these steps:

1. Open the Worksheet containing the table you want to filter.

2. Select the query you wish to use as a subquery in the condition. Drag it from the asset tree to an empty cell on the Worksheet. This creates a new table with the name of the query.



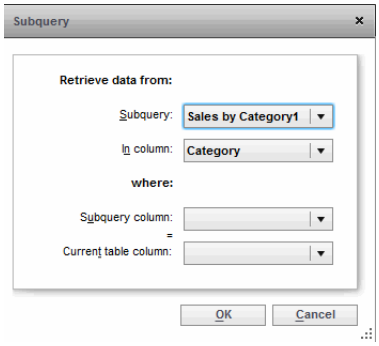
3. Click the 'Condition' button on the title bar of the table you wish to filter. This opens the 'Condition' dialog box.

4. Open the edit field in either the 'Simple Condition' mode (click the 'More' button) or 'Advanced Condition' mode (click the pre-aggregate or post-aggregate 'Edit' button).

5. Select 'Subquery' from the rightmost popup menu. This displays the 'Subquery' button.



6. Click the ‘Subquery’ button. This opens the ‘Subquery’ dialog box.



7. In the ‘Subquery’ menu, specify the desired subquery table. (This is the table created when you dragged the query to the Worksheet.)

8. Select options in the lower three popup menus, as needed.

   You may not need to make every menu selection; this will depend on how you want to use the subquery result set. The ‘In column’ menu specifies the subquery column that contains the result set to use in the condition. The bottom two menus control table synchronization for row-dependent conditions.

9. Click ‘OK’ to close the ‘Subquery’ dialog box.

10. After you have specified the subquery to use, define the rest of the condition as described earlier.

See Also

Independent Query, in *Advanced Data Modeling*, for information on queries.

Filtering, for details on specifying filtering conditions.

Specifying a Subquery, for more information on defining the subquery.

5.2.9 Specifying a Subquery

You may not need to specify every setting in the ‘Subquery’ dialog box. The required menu choices depend on how you wish to use the subquery result set. This section explains three different cases for using a subquery.

Subquery Returns a Fixed Value

If the subquery returns a single fixed value (i.e., the subquery table in the Worksheet contains a single cell), then in the 'Subquery' dialog box you only need to specify the *name* of the subquery. No further settings are required.

Subquery Returns a Fixed List

If you want to use a particular column from the subquery as a fixed list in the condition, you need to specify the query name in the 'Subquery' dialog box, and then select the desired subquery column from the 'In column' menu. A subquery that returns a column is typically used in conjunction with the 'one of' condition clause to filter values that are members (or non-members) of the returned list. You do not need to make selections in the 'Subquery column' and 'Current table column' menus.

Example: Subquery Returns a Fixed List

Assume that you have a table of companies that purchased a 'Wireless Mouse' product, and a second table of companies that purchased a 'Fast Go Game' product. You would like to filter the 'Wireless Mouse' table to show only the companies that *did not* purchased the 'Fast Go Game.' To do this, define a condition on the 'Wireless Mouse' table that uses the 'Fast Go Game' table as a subquery. Follow the steps below:



1. Create a new Worksheet.
2. Drag the 'Wireless Mouse' query on to an empty cell in the Worksheet. This creates a new table named 'Wireless Mouse1'.
3. Drag the 'Fast Go Game' query on to an empty cell in the Worksheet. This creates a new table named 'Fast Go Game1'.
4. In the 'Fast Go Game1' table, rename the 'Customers' column to 'Customers_FGG'. This will make the next steps more clear.



5. Click on the 'Condition' button located in the upper right corner of the 'Wireless Mouse1' table's title bar. This will open the 'Conditions' dialog box.
6. Deselect the 'Advanced Conditions' checkbox if it is selected.
7. Click on the 'More' button to open the 'Edit' panel.
8. Using the menus in the 'Edit' panel, specify the following expression:

```
[Customers][is not][one of]
```

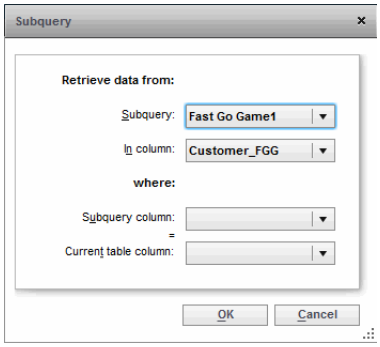
9. Click on the button with the triangle, located on the far right, and select ‘Subquery’. The value text box is replaced with the ‘Subquery’ button.



10. Click the ‘Subquery’ button. This opens the ‘Subquery’ dialog box.

11. Select ‘Fast Go Game1’ from the ‘Subquery’ menu. This is the name of the subquery table to use.

12. Select ‘Customers_FGG’ from the ‘In column’ menu. This is the column of the subquery which contains the list of values to use in the condition expression.



13. Click ‘OK’ to close the ‘Subquery’ dialog box. (You do not need to make selections from the bottom two menus.)

14. Click ‘OK’ to exit the ‘Condition’ dialog box.

15. To view the results, press the ‘Change View’ button in the title bar of each table and select ‘Live Preview’ from the menu.



The ‘Wireless Mouse1’ table has now been filtered to display only the companies that did *not* purchase the ‘Fast Go Game’. Note that the two tables are now graphically linked together by an arrow to indicate that the

'Fast Go Game1' table provides a subquery result to the 'Wireless Mouse1' table.

The usefulness of defining the filter based on a subquery (rather than hard-coding the list of companies) is that updates to the underlying database which cause the 'Fast Go Game' query to return a different result set will be automatically reflected in the filtered 'Wireless Mouse' table as well.

¢

Subquery Returns a Row-Dependent Value

A subquery can return a value that is dependent on the *row* of the main table. Such a row-dependent subquery is useful when you want to filter rows of the main table based on a comparison with corresponding rows of another table (i.e., the subquery table).

For example, consider a case where one table displays sales by year for NY, and a second table displays sales by year for CA. You want the NY table to display records *only* for those years where sales were greater than sales in CA. For instance, you want the NY table to display the record for year 2002 *only if* sales in NY for 2002 were greater than sales in CA for 2002.

To configure a row-dependent subquery, follow these steps:

1. Select the table you wish to filter.

2. Open the 'Condition' dialog box, and then open the 'Subquery' dialog box. See the *Filtering* and *Using a Subquery in a Condition* sections for detailed instructions on how to do this.

3. Select the subquery name in the 'Subquery' menu. (In the example above, the subquery is the CA sales table.)

4. Select the desired subquery column from the 'In column' menu. This is the column of the subquery containing the actual values to be tested in the condition. (In the example above, this would be the 'sales' column of the CA table, since this column contains the values that the condition will test.)

5. From the 'Subquery column' menu, select the column in the *subquery table* that indexes the subquery condition column (selected in the previous step). The 'Subquery column' column is the subquery join column; it will be used to "look up" the condition values corresponding to a particular row in the main table. (In the example above, the 'Subquery column' column would be the 'year' column of the CA table, since this is the column that indexes the 'sales' column of the CA table.)

6. From the ‘Current table column’ menu, select the column in the *main table* that contains the indexing values corresponding to those in the ‘Subquery column’ of the subquery table. (In the example above, the ‘Current table column’ would be the ‘year’ column of the NY table, since this is the column that corresponds to the CA ‘year’ column.)

The ‘Subquery column’ and ‘Current table column’ serve to connect the subquery table and main table. For each row in the main table, the value in the ‘Current table column’ is matched against the values in the ‘Subquery column’. The results returned by the subquery for each row are the values of the condition ‘In column’ in the rows where the ‘Subquery column’ value is matched.

Example: Subquery Returns Row-Dependent Value

This example illustrates how to use a row-dependent subquery. Suppose you have the total sales information for all product categories, both for this year and last year. However, you want to display only those product categories that have yielded more sales revenue this year than last year. To do this, you can use last year’s total sales as a subquery to provide the data for the condition you place on this year’s sales table.



1. Create a new Worksheet.

2. Drag the ‘Sales this year’ query from the ‘Orders’ > ‘DWS’ folder to an empty cell in the Worksheet. This creates a new table named ‘Sales this year1’.

3. Drag the ‘Sales last year’ query from the same folder to an empty cell in the Worksheet. This creates a new table named ‘Sales last year1’.

4. Rename the columns in the ‘Sales last year1’ table: Rename ‘Category’ to ‘Category_LY’, and rename ‘Total Sales’ to ‘Total_Sales_LY’. This will make the following steps more clear.



5. Click on the ‘Condition’ button located in the upper right corner of the ‘Sales this year1’ table’s title bar. This will open the ‘Conditions’ dialog.

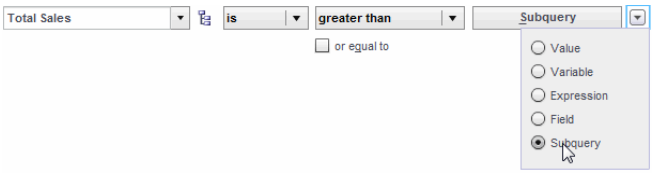
6. Deselect the ‘Advanced Conditions’ checkbox if it is selected.

7. Click on the ‘More’ button to open the ‘Edit’ panel.

8. Using the menus in the ‘Edit’ panel, specify the following expression:

```
[Total Sales][is][greater than]
```

9. Click on the button with the triangle pointing down, located on the far right, and select ‘Subquery’. The value text box is replaced with the ‘Subquery’ button.



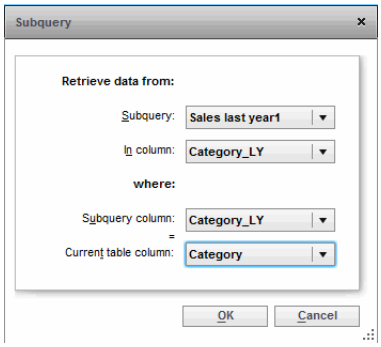
10. Click on the ‘Subquery’ button. This opens the ‘Subquery’ dialog box.

11. Select ‘Sales last year1’ from the ‘Subquery’ menu.

12. Select ‘Total_Sales_LY’ from the ‘In column’ menu. This is the column of the subquery containing the values to be used in the condition test.

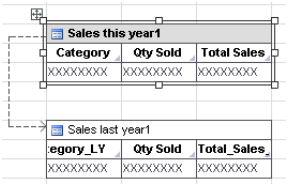
13. Select ‘Category_LY’ from the ‘Subquery column’ column. This is the subquery column that will be matched up against the ‘Current table column’ in the main table. (It is the “join column” of the subquery table.)

14. Select ‘Category’ from the ‘Current table column’ menu. This is the main table column that will be matched up against the ‘Subquery column’ in the subquery table. (It is the “join column” of the main table.)



15. Click the ‘OK’ button to close the ‘Subquery’ dialog box.

16. Click the ‘OK’ button in the ‘Condition’ dialog box. The two tables are now shown with a graphical link, indicating that the ‘Sales this year1’ table relies on subquery data from the ‘Sales last year1’ table.

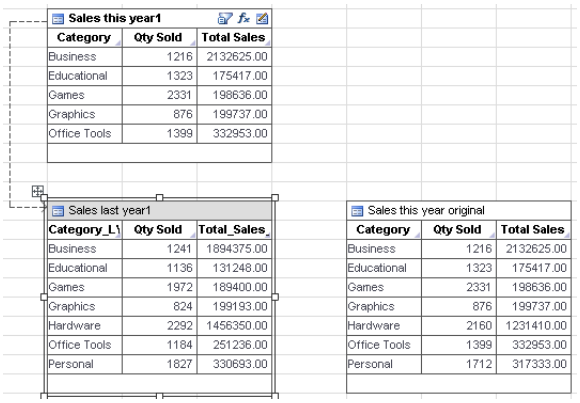


17. For each table, press the ‘Change View’ button in the title bar of the table and select ‘Live Preview’ from the menu.

18. Expand the table borders as needed to view the results.

19. (Optional) Compare the filtered ‘Sales this year1’ table to the original ‘Sales this year1’ table by dragging the ‘Sales this year’ query (again) from the asset tree to an empty Worksheet cell. Rename it ‘Sales this year original’.

Note that the filtered ‘Sales this year1’ table now contains only the categories for which sales were higher than the previous year.



Sales this year1

Category	Qty Sold	Total Sales
Business	1216	2132625.00
Educational	1323	175417.00
Games	2331	198636.00
Graphics	876	199737.00
Office Tools	1399	332953.00

Sales last year1

Category_LY	Qty Sold	Total_Sales
Business	1241	1894375.00
Educational	1136	131248.00
Games	1972	189400.00
Graphics	824	199193.00
Hardware	2292	1456350.00
Office Tools	1184	251236.00
Personal	1827	330693.00

Sales this year original

Category	Qty Sold	Total Sales
Business	1216	2132625.00
Educational	1323	175417.00
Games	2331	198636.00
Graphics	876	199737.00
Hardware	2160	1231410.00
Office Tools	1399	332953.00
Personal	1712	317333.00

¢

5.3 Grouping

The ‘Group and Aggregate’ button in a table’s title bar allows you to create groupings in the table.

5.3.1 Grouping and Aggregating Data

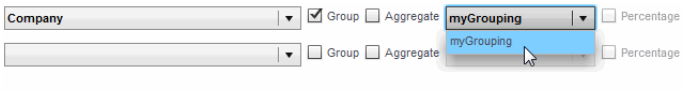
To define grouping and aggregation columns (summary columns) for a Data table, follow these steps:

1. Press the ‘Group and Aggregate’ button in the table’s title bar. This opens the ‘Group and Aggregate’ dialog box.

2. Select the grouping columns by clicking the ‘Group’ check box next to one or more column names.

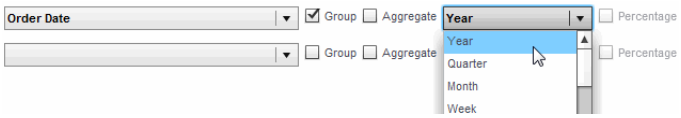
When you enable grouping for a column, by default each distinct value in the column defines its own group. Aggregation is then based on these groups. For example, if you group a column of state names, each distinct state name will define its own group, and aggregate values will be computed for each of those groups.

3. (Optional) If a user-defined group matches the grouping column data format (String, Double, etc.), you can select the user-defined group from the right-side menu.



User-defined groups give you greater flexibility in partitioning the entries in a column for grouping. If you select a user-defined grouping from the menu, aggregations will be based on that user-defined grouping.

4. (Optional) If you specify grouping for a column that has a Date format, you can select a predefined or user-defined Date grouping (Year, Quarter, etc.) from the right-side menu.



The actual date values in the grouping column will be replaced with the new group labels, e.g., ‘1st Quarter’, ‘January’, etc. If the dates are already grouped by the database, select ‘None’.

5. Click the ‘Aggregate’ check box for columns containing the measures that you wish to aggregate (summarize).

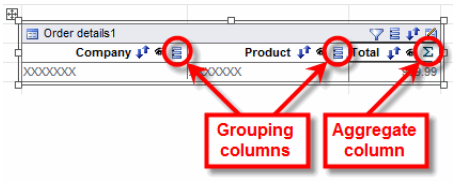
6. For the aggregate columns, choose the method of aggregation (Sum, Correlation, etc.).

7. (Optional) For the univariate aggregation methods (Sum, Max, etc.), select the ‘Percentage’ check box to display the aggregated measure as a percentage of the grand total.

8. (Optional) For the bivariate aggregation methods (Correlation, etc.), select the second operand (column) for the computation from the ‘with’ menu.

9. Press ‘OK’ to close the ‘Group and Aggregate’ dialog box.

The table is now grouped. Only the table columns that are either grouping columns or aggregate columns are shown in the grouped table. All other columns are hidden. In the grouped table, each column has an icon indicating whether it is a grouping column or an aggregate column. Click any of these icons to reopen the 'Group and Aggregate' dialog box.



See Also

Creating a Named Grouping, for more information on user-defined groupings.

Appendix C.2, *Built-in Date Groupings*, for a list of predefined date groupings.

Crosstab Aggregation Measures, for more details about aggregation methods.

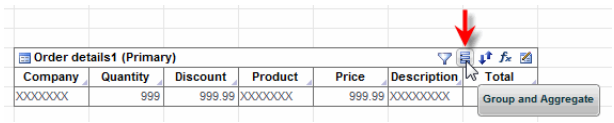
Switching Between Detail and Aggregate View, to revert back to ungrouped table.

Creating a Numeric Range Column, to create a new grouping of existing column.

Example: Grouping and Aggregating Data

Assume you have order information for several companies and are interested in seeing how much revenue was generated by each of the individual products sold. To do this, group the table data by 'Product' and summarize by 'Total'. Follow the steps below:

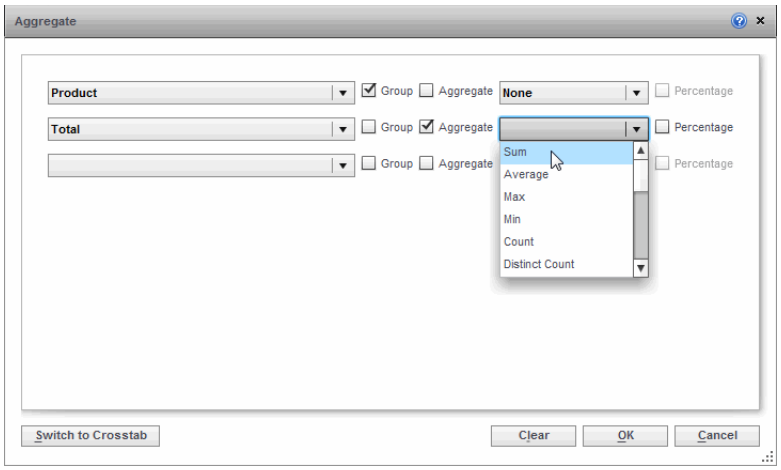
1. Create a new Worksheet.
2. Expand the 'Data Source' node, and the 'Orders' node.
3. Drag the 'Order details' query on to an empty cell in the Worksheet. This creates a new table named 'Order details1'.
4. Press the 'Group and Aggregate' button in the table title bar.



This opens the 'Group and Aggregate' dialog box.

5. Select 'Group' for the 'Product' column and 'Aggregate' for the 'Total' column.

6. Set the aggregate function for the ‘Total’ column to ‘Sum’.



7. Click on the ‘OK’ button to apply the grouping and summarization.

8. Preview the table. Notice that only the ‘Product’ and ‘Total’ fields are included in the table and the columns not included in the grouping or aggregation have been removed.

	A	B
1	Order details1	
2	Product	Total
3	17 Inch LCD	373936.50
4	19 inch LCD	497007.50
5	Animal World	104158.60
6	Barbie's Fash	156542.45
7	Combat Hero	44001.70
8	Easy Chess	53518.50
9	Fancy Menus	96860.75
10	Fast Go Game	304260.00
11	Fast Mail	67666.95
12	Green Planet	115072.50

¢

5.3.2 Defining a Crosstab Table

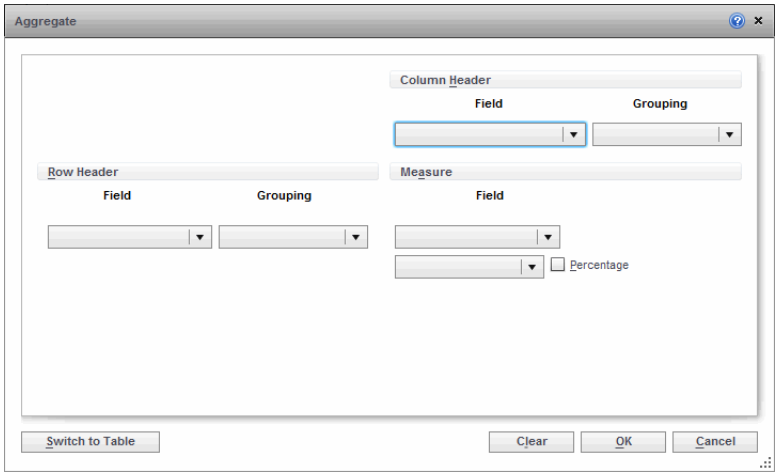
A crosstab table is also known as a pivot table. A Worksheet crosstab contains one column header, one or more row headers, and one measure.

The values at the row-column intersections of the crosstab table represent summary (*aggregate*) information of the measure. For example, a crosstab with row headers representing ‘Salesperson’ and column headers representing ‘Month of Year’ might contain values at the intersections representing ‘Average Sale Price’ or ‘Maximum Sale Price’. The measure in both cases is ‘Sale Price’. The summarization method is, respectively, Average or Maximum. By using summarization, crosstab tables display large amounts of data in a compact form.

To create a crosstab table from an existing Data Table, follow these steps.

1. Press the ‘Group and Aggregate’ button in the table title bar. This opens the ‘Group and Aggregate’ dialog box.

2. In the ‘Group and Aggregate’ dialog box, click ‘Switch to Crosstab’. The ‘Group and Aggregate’ dialog box switches to the crosstab view.



3. In the ‘Row Header’ panel, select one or more fields from the left menus.

4. In the ‘Column Header’ panel, select a field from the left menu.

5. (Optional) For each selected field in the ‘Row Header’ and ‘Column Header’ panels, select a Named Grouping (if one exists) from the corresponding right menu.

6. In the ‘Aggregate’ panel, select a measure column from the left menu. This is the column whose values will be summarized.

7. In the ‘Aggregate’ panel, select the aggregation method from the right menu.

   If you select a bivariate aggregation measure (e.g., ‘Correlation’, ‘Weighted Average’, etc.), select the second operand (column) from the ‘with’ menu.

8. (Optional) To display the measure as a percentage of the grant total, select the ‘Percentage’ option.

9. Click ‘OK’ to create the crosstab table.

See Also

Creating a Named Grouping, for more information on user-defined groupings.

Crosstab Aggregation Measures, for more details about aggregation methods.
Switching Between Detail and Aggregate View, to revert back to ungrouped table.

5.3.3 Crosstab Aggregation Measures

The table below shows the crosstab aggregation measures available for table grouping.

AGGREGATE MEASURE	DESCRIPTION
Sum	Displays the sum of the measure values for the given row and column headings
Average	Displays the average of the measure values for the given row and column headings
Max	Displays the maximum of the measure values for the given row and column headings
Min	Displays the minimum of the measure values for the given row and column headings
Count	Displays the total count of measure values for the given row and column headings. This represents the total number of records corresponding to the given row and column headings, and is the same value for any selected measure.
Distinct Count	Displays the count of unique measure values for the given row and column headings.
First	Displays the first value for the measure (for the given row and column headings) when sorted based on the values in a second column, specified by the menu labeled ‘by’.
Last	Displays the last value for the measure (for the given row and column headings) when sorted based on the values in a second column, specified by the menu labeled ‘by’.
Correlation	Displays the Pearson correlation coefficient for the correlation between the measure values (for the given row and column headings) and the corresponding values in a second column, specified by the menu labeled ‘with’.
Covariance	Displays the covariance between the measure values (for the given row and column headings) and the corresponding values in a second column, specified by the menu labeled ‘with’.
Variance	Displays the (sample) variance of the measure values for the given row and column headings.
Std Deviation	Displays the (sample) standard deviation of the measure values for the given row and column headings.

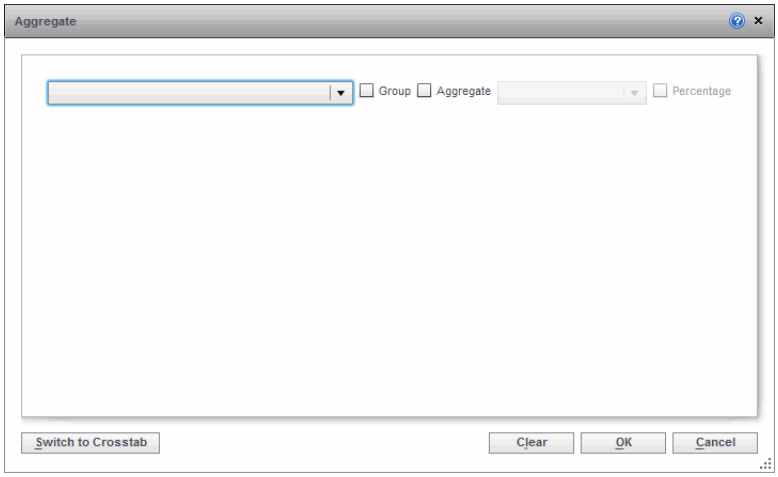
AGGREGATE MEASURE	DESCRIPTION
Variance (Pop)	Displays the (population) variance of the measure values for the given row and column headings.
Std Deviation (Pop)	Displays the (population) standard deviation of the measure values for the given row and column headings.
Weighted Average	Displays the weighted average of the measure values for the given row and column headings. The weights are given by the corresponding values in a second column, which is specified by the menu labeled 'with'.

Example: Crosstab Aggregation

Suppose that you want to know the total amount of money spent by each customer on each of the products. Follow the steps below to display that information in a crosstab table:



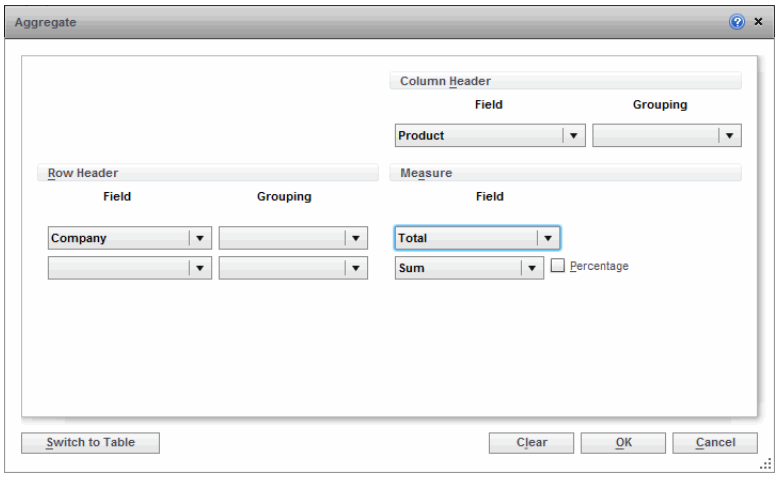
1. Create a new Worksheet.
2. Expand the 'Data Source' node, and the 'Orders' node.
3. Drag the 'Order details' query on to an empty cell in the Worksheet. This creates a new table named 'Order details1'.
4. Press the 'Group and Aggregate' button in the table's title bar. This opens the 'Group and Aggregate' dialog box.



5. Click on the 'Switch to Crosstab' button. The view changes to enable you to specify the column headers and the summary column.
6. Select 'Company' from the first drop-down list under the 'Row Header'.

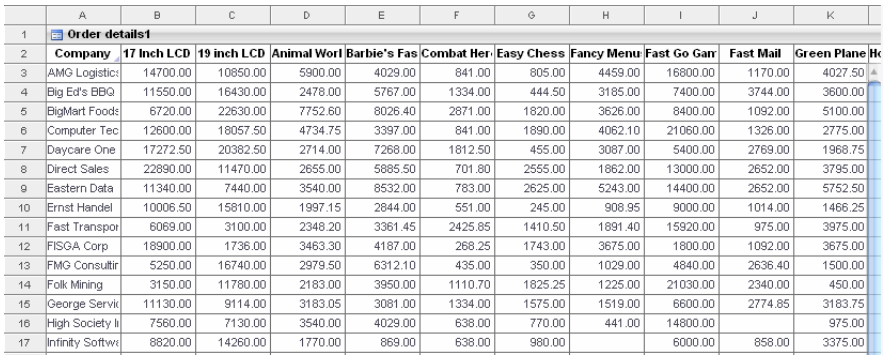
7. Select ‘Product’ from the ‘Column Header’ drop-down list.

8. Select ‘Total’ from the ‘Measure’ drop-down list and ensure that the aggregate function is set to ‘Sum’.



9. Click on the ‘OK’ button to save the changes.

10. Preview the table.



	A	B	C	D	E	F	G	H	I	J	K
1	Order details1										
2	Company	17 Inch LCD	19 inch LCD	Animal Worl	Barbie's Fas	Combat Her	Easy Chess	Fancy Menu	Fast Go Gan	Fast Mail	Green Plane
3	AMG Logistics	14700.00	10850.00	5900.00	4029.00	841.00	805.00	4459.00	16800.00	1170.00	4027.50
4	Big Ed's BBQ	11550.00	16430.00	2478.00	5767.00	1334.00	444.50	3185.00	7400.00	3744.00	3600.00
5	BigMart Foods	6720.00	22630.00	7752.60	8026.40	2871.00	1820.00	3626.00	8400.00	1092.00	5100.00
6	Computer Tec	12600.00	18057.50	4734.75	3397.00	841.00	1890.00	4062.10	21060.00	1326.00	2775.00
7	Daycare One	17272.50	20382.50	2714.00	7268.00	1812.50	455.00	3087.00	5400.00	2769.00	1968.75
8	Direct Sales	22890.00	11470.00	2655.00	5885.50	701.80	2555.00	1862.00	13000.00	2652.00	3795.00
9	Eastern Data	11340.00	7440.00	3540.00	8532.00	783.00	2625.00	5243.00	14400.00	2652.00	5752.50
10	Ernst Handel	10006.50	15810.00	1997.15	2844.00	551.00	245.00	908.95	9000.00	1014.00	1466.25
11	Fast Transpor	6069.00	3100.00	2348.20	3361.45	2425.85	1410.50	1891.40	15920.00	975.00	3975.00
12	FISGA Corp	18900.00	1736.00	3463.30	4187.00	268.25	1743.00	3675.00	1800.00	1092.00	3675.00
13	FMG Consultir	5250.00	16740.00	2979.50	6312.10	435.00	350.00	1029.00	4840.00	2636.40	1500.00
14	Folk Mining	3150.00	11780.00	2183.00	3950.00	1110.70	1825.25	1225.00	21030.00	2340.00	450.00
15	George Servic	11130.00	9114.00	3183.05	3081.00	1334.00	1575.00	1519.00	6600.00	2774.85	3183.75
16	High Society I	7560.00	7130.00	3540.00	4029.00	638.00	770.00	441.00	14800.00		975.00
17	Infinity Softw	8820.00	14260.00	1770.00	869.00	638.00	980.00		6000.00	858.00	3375.00

¢

5.3.4 Switching Between Detail and Aggregate View

When you define grouping or aggregation on a Data Table, the resulting table is automatically placed in “aggregate view,” which shows only the grouping and aggregation columns.

- To switch a table from aggregate view to detail view (displaying all columns, without grouping and aggregation), click the ‘To detail’ button on the table title bar.

- To switch a table from detail view to aggregate view (hiding all non-aggregate and non-grouping columns), click the 'To Aggregate' button on the table title bar.

Example: Switching Between Detail and Aggregate View

The following example illustrates how to toggle from detail to aggregate view.

1. Create a new Worksheet.
2. Expand the 'Data Source' node, and the 'Orders' node.
3. Drag the 'Order details' query on to an empty cell in the Worksheet. This creates a new table named 'Order details1'.
4. Click the 'Group and Aggregate' button in the table title bar. This opens the 'Group and Aggregate' dialog box.
5. Select the 'Group' option for the 'Product' column and 'Aggregate' for the 'Total' column.
6. Ensure that the aggregate function for the 'Total' column is set to 'Sum'.
7. Click on the 'OK' button to apply the grouping and summarization.
8. Press the 'Change View' button in the table title bar, and select 'Live Preview' from the menu.

   Notice that since grouping and summarization is specified for this table, only the summarized data is displayed in the table, and all of the columns that are not included in the grouping or summarization have been removed.



Order details1 (Prima	
Product	Total
17 Inch LCD	373936.50
19 inch LCD	497007.50
Animal World	104158.60
Barbie's Fash	156542.45
Combat Hero	44001.70
Easy Chess	53518.50
Fancy Menus	96860.75
Fast Go Game	304260.00
Fast Mail	67666.95
Green Planet	115072.50
House Hunter	229994.1

9. To switch to the detail view, press the 'Change View' button in the title bar and select 'Full' (meta-data) or 'Detail Preview' (live data). Notice that all of the columns are now displayed in the table, but the group and aggregate columns are displayed with darker headings.

Order details1

Company	Quantity	Discount	Product	Price	Description	Total
Interstate Sho	6	0.00	NetStorage	2850.00	500 GB netwo	17100.00
Interstate Sho	5	0.00	Animal World	59.00	Educational sc	295.00
Interstate Sho	1	0.00	Mega Icons	550.00	Over 10,000 i	550.00
Interstate Sho	16	0.00	WebCalendar	248.00	Web enabled	3968.00
Interstate Sho	18	0.00	19 inch LCD	310.00	19 inch LCD m	5580.00
Interstate Sho	18	0.00	Sync Me	145.00	Palm power s	2610.00
Interstate Sho	15	0.00	Math for Me	235.00	Math tutor	3525.00
Interstate Sho	16	0.00	Wireless Mou	40.00	Wireless optic	640.00
Interstate Sho	1	0.00	NetStorage	2850.00	500 GB netwo	2850.00

10. To switch back to the summarized view of the data, press the ‘Change View’ button in the title bar and select ‘Default’ (meta-data) or ‘Live Preview’ (live data).

¢

6 Creating Non-Tabular Assets

Non-tabular assets include Named Groupings and Variables. Like tables, they are reusable across multiple Worksheets and reports. This section explains how to create each of these asset types.

For more information about working with assets and the Asset Repository, see *Assets: Reusable Worksheet Components*.

6.1 Creating a Named Grouping

The default grouping procedure simply partitions a column based on its distinct values. For example, if a column contains state abbreviations CA, NY, and WA, then the default grouping produces one group for CA, one group for NY, and one group for WA.

If this default grouping is not satisfactory, you can create a different partition with Named Grouping. A Named (User-Defined) Grouping is a reusable set of custom conditions (i.e., rules) that partition the data into the desired groups.

To create a Named Grouping, follow these steps:



1. Click the 'New Object' button in the toolbar, and select 'Grouping'. This opens the 'Grouping' dialog box.

2. In the 'Name' field, enter a name for the grouping assembly. This is the title of the grouping object in the Worksheet. (After the grouping object is created, you can edit the grouping name from the text field at the top of the Worksheet.)

3. If you want the grouping to be accessible to all columns of a certain data type, select the desired data type from the 'Type' menu.

   For example, you could use the 'starting with' operator to define a grouping based on initial letter (i.e., A, B, C, etc.). If you specified type 'String' for this grouping, you could apply the grouping to any column of that type in a Worksheet or report. This would allow you to reuse the same alphabetical grouping for customer names, company names, product descriptions, and any other column that has 'String' type data.

4. If you want the grouping to be applicable only to a particular column or attribute of a particular data source, select the desired data source from the 'Only for' menu. Select the desired column from the 'Attribute' menu.

You can select a column or attribute from any accessible data source, not restricted to those in the current Worksheet. The grouping defined for this column or attribute will then be available wherever the column or attribute is used in a Worksheet or report.

You can now specify the conditions that define membership in each group of the User defined Grouping.

5. Click the 'Add' button to add a new group. This opens the 'Add Condition List' dialog box. (This is the same as the dialog box used for adding advanced conditions to tables.)

6. In the 'Group Name' field, enter the name of the group you wish to define. (For example, "East Coast states".)

7. Enter the conditions that define membership in this particular group, and click 'OK'.

8. Repeat the above three steps for each group you wish to add to the grouping:

9. In the 'Grouping' dialog box, choose 'Group all others together' to create a default group called "Others" that will contain all column values that do not satisfy any of the specified group conditions. Otherwise, select 'Leave others in their own group'.

10. Click 'OK' to exit the 'Grouping' dialog box.

The Named Grouping is now shown in the Worksheet as a table whose title is the group name, and whose rows are individual group names. (You can drag the bottom border of the table to see all the group names.)

See Also
Grouping, for information on default grouping operations.
Defining Advanced Conditions, for information on specifying conditions.
Saving an Asset, to save the defined grouping to the Repository.
Using a Named Grouping, to use the grouping in other Worksheets and reports.

Example: Creating a Named Grouping

Suppose you want to create a group for all the states on the west coast, so that every time the customer data is displayed in either a table, chart, or section element in the report, all of the customers from the west coast states are grouped together.

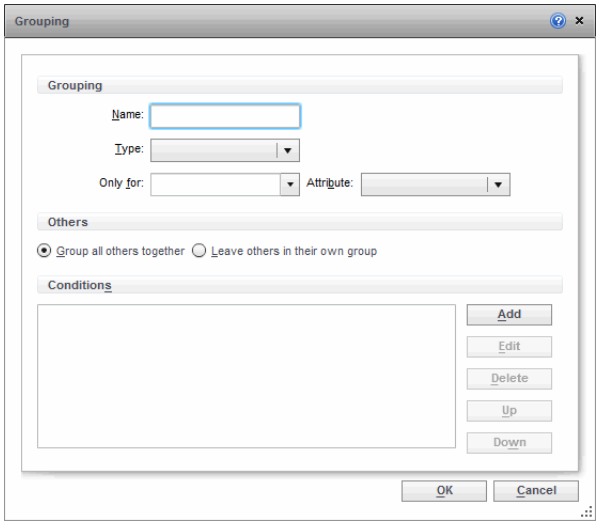
Follow these steps to create the 'WestCoast' group.



1. Create a new Worksheet.



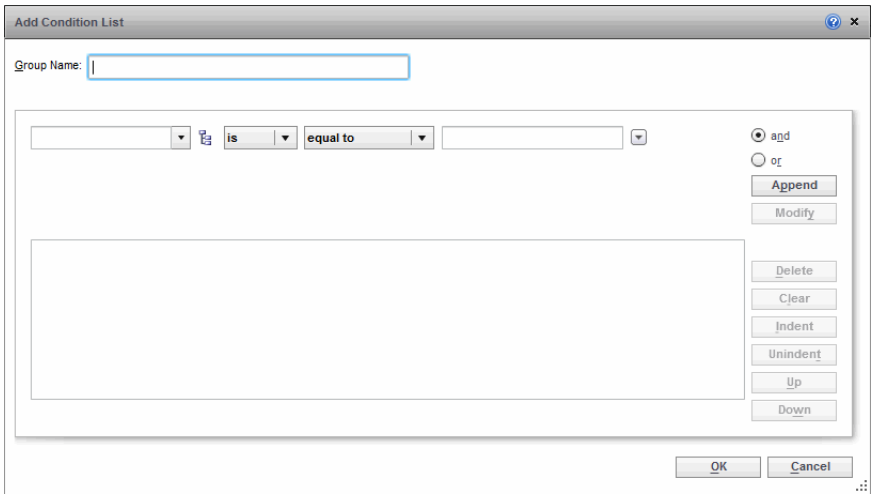
2. Click on the ‘New Object’ button, and select the ‘Grouping’ option. This opens the ‘Grouping’ dialog box.



3. Specify “WestCoast” as the name.

4. For the ‘Type’ select ‘String’.

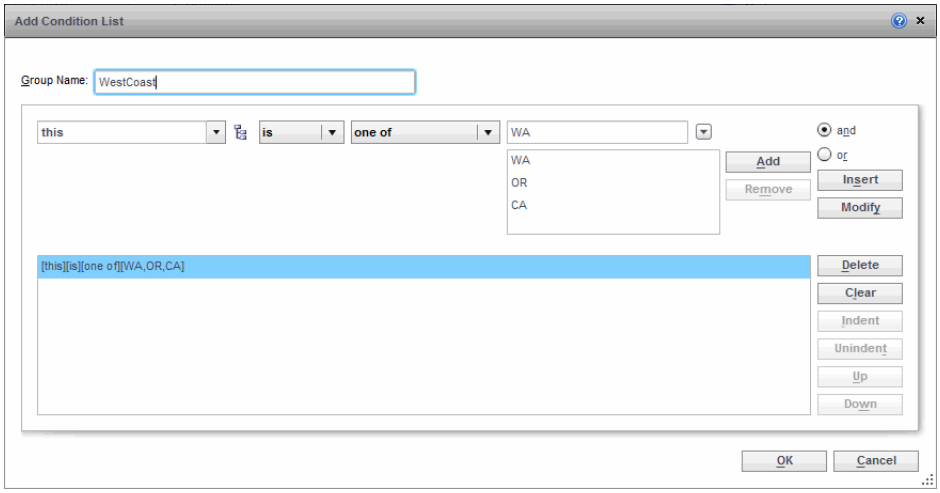
5. Click on the ‘Add’ button to specify the conditions which the data has to satisfy in order to qualify to be a part of this new group. This opens the ‘Add Condition List’ dialog.



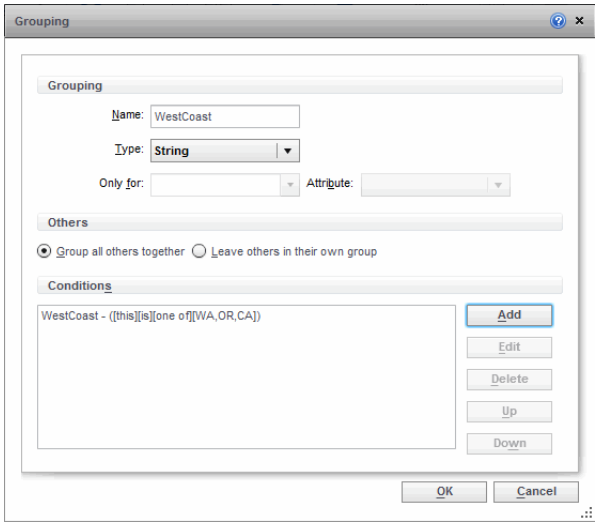
6. Specify ‘WestCoast’ as the Group Name.

7. Specify the following condition, and click the ‘Append’ button.

```
[this][is][one of][WA,OR,CA]
```

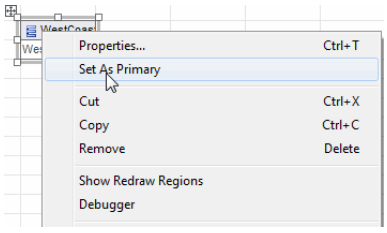


8. Click on the ‘OK’ button to exit the ‘Add Condition List’ dialog and go back to the ‘Group’ dialog.



9. Click on the ‘OK’ button.

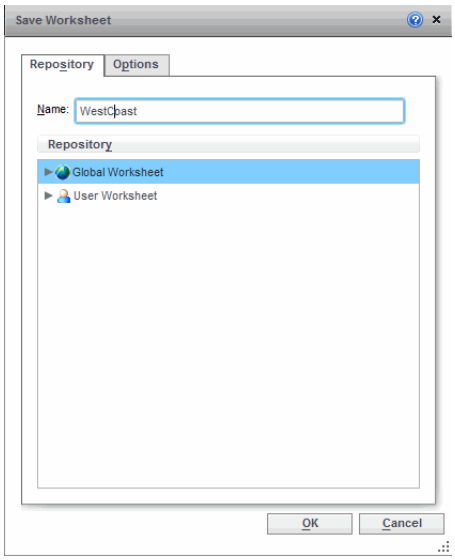
10. Right-click on the title for the ‘WestCoast’ group and select ‘Set as Primary’.



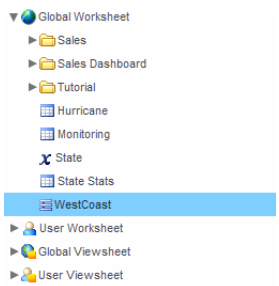


11. Click on the ‘Save’ button, located in the top toolbar. The ‘Save as Worksheet’ dialog appears.

12. Specify “WestCoast” as the name of the asset and select ‘Global Worksheet’ as its scope.



13. Click on the ‘OK’ button to save the newly created user-defined grouping. Notice that the name of the grouping on the asset tree is prepended with a ‘Grouping’ icon, indicating that the asset is of the ‘Named Group’ type.



¢

6.2 Creating a Variable

Variables provide a way to prompt the user to enter specific filtering conditions for tables in the Worksheet. You can use Variables in conditions on tables and Named Groups.

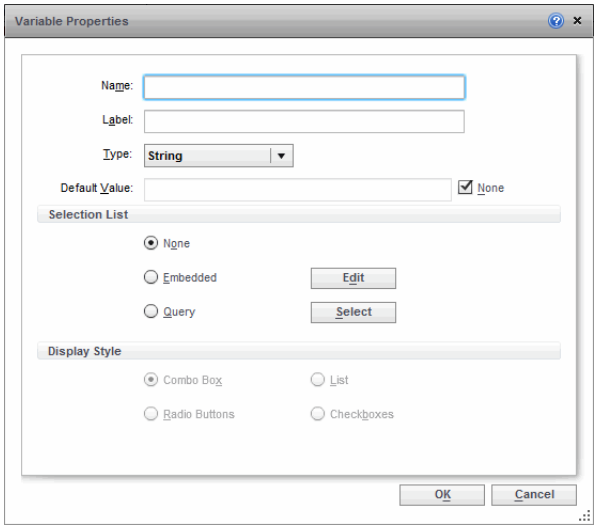
See *Saving an Asset* and *Using a Variable* for more information about working with Variables.

6.2.1 Defining a Variable

To create a new Variable, follow these steps:



1. Click the ‘New Object’ button in the top toolbar, and select ‘Variable’. This opens the ‘Variable Properties’ dialog box.



2. Enter a ‘Name’ for the new Variable asset. This is the name that appears in the title bar of the Variable asset.

3. Enter a ‘Label’ for the Variable. This is the label that appears in the parameter dialog box that prompts the user.

4. Select the data type of the Variable from the ‘Type’ menu. This is the data type of the values that the Variable will provide to the filtering condition.

5. In the ‘Selection List’ panel, specify the source of the Variable data:

   a. Select ‘None’ to prompt the user with an empty text field (no choices).

   b. Select ‘Embedded’ to enter a fixed list of choices.

   c. Select ‘Query’ to populate the list of choices with the data in an existing table column.

6. In the ‘Display Style’ panel, select the type of list to display to the user.

   The ‘Combo Box’ and ‘Radio Buttons’ options allow the user to make a single selection. The ‘List’ and ‘Checkboxes’ options allow

the user to make multiple selections. (Multiple selections can be used in conjunction with the ‘one of’ condition operator.)

7. Click ‘OK’ to exit the dialog box.

See Also

Creating an Embedded Variable List, to manually specify the Variable values.

Creating a Query-Based Variable List, to automatically generate Variable values.

Example: Defining a Variable

Say you have multiple Data Worksheets including the order and contact information pertaining to all customers. It would be helpful to have a Variable that provided a specific state value in order to filter the data so that only information pertaining to the customers in the specified state was included in the data analysis.

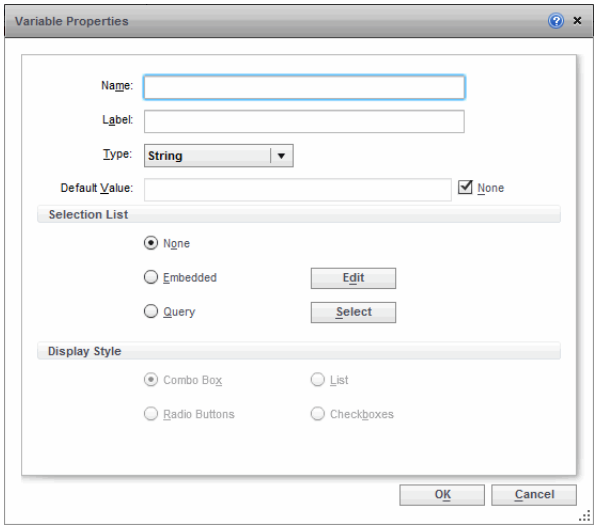
Instead of creating the ‘state’ Variable for all of the Data Worksheets, you can create it once and save it as a ‘Variable’ asset in the Asset Repository. Then, you can reuse the Variable in all of the relevant Data Worksheets. Follow these steps to create a Variable asset:



1. Create a new Worksheet.



2. Click on the ‘New Object’ button located in the top toolbar, and select the ‘Variable’ option. This opens the ‘Variable Properties’ dialog box.



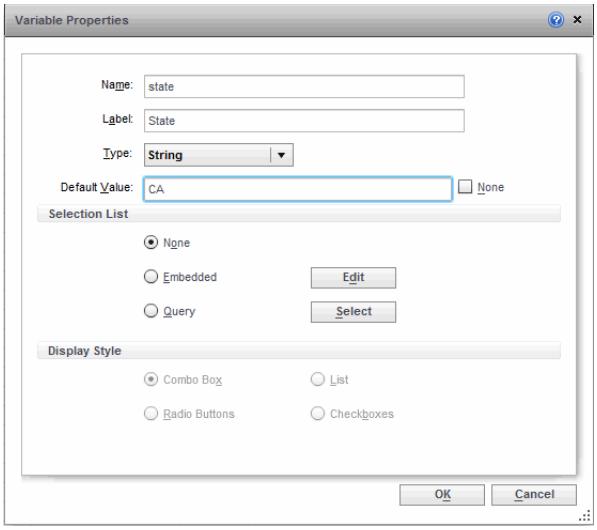
3. Specify “state” as the name of the Variable.

4. Specify “State” as the Label.

5. Select ‘String’ from the drop-down list for Type.

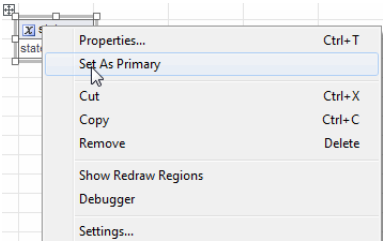
6. Uncheck the ‘None’ check box next to ‘Default Value’, and enter “CA” as the default value.

7. Ensure that ‘None’ is selected for the Selection List.



8. Click on the ‘OK’ button

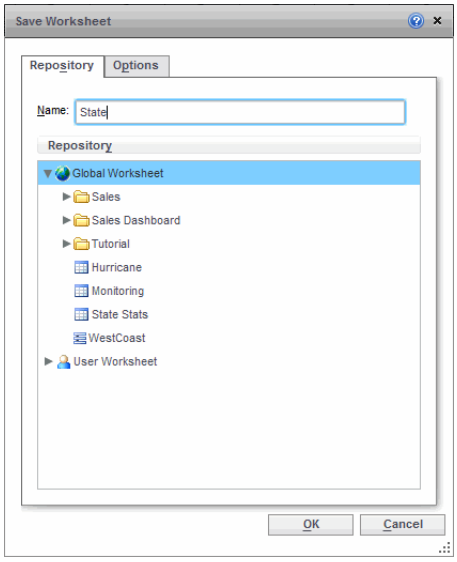
9. Right-click on the Variable’s title and select ‘Set as Primary’.



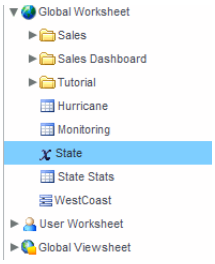


10. Click on the ‘Save’ button in the top toolbar. This opens the ‘Save as Worksheet’ dialog.

11. Specify “State” as the asset name and select ‘Global Worksheet’ as its scope.



12. Click on the ‘OK’ button to add the ‘State’ Variable to the Asset Repository. Notice that the name of the asset on the asset tree is prepended with a Variable icon indicating that the asset is of the ‘Variable’ type.



This example is continued in *Using a Variable*, where the ‘state’ Variable is used in another Worksheet.

¢

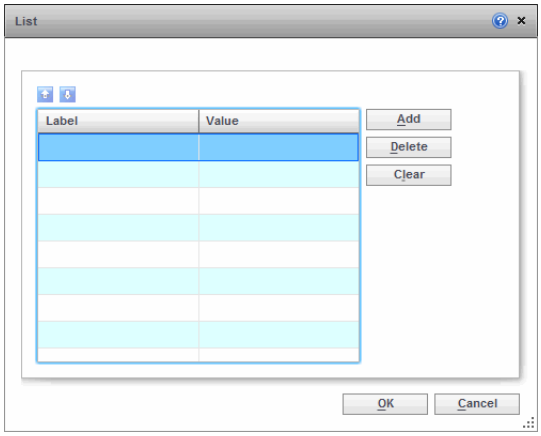
6.2.2 Creating an Embedded Variable List

To create an embedded list for a Variable, follow these steps:

1. Right-click the Variable asset, and select the ‘Properties’ option from the context menu. This opens the ‘Variable Properties’ dialog box.

2. In the ‘Selection List’ panel of the ‘Variable Properties’ dialog box, select the ‘Embedded’ option.

3. Click the ‘Edit’ button to open the ‘List’ dialog box.



4. Click ‘Add’ to create a new label/value pair. A label/value pair defines a particular choice that will be shown to the user in the input dialog box.

5. In the ‘Label’ field, enter the text that should be displayed to the user for this particular choice.

6. In the corresponding ‘Value’ field, enter the text that should be used in the filtering condition for this particular choice. (If the ‘Type’ option in the ‘Variable Properties’ dialog box specifies a numeric type, you can only enter numerical values.)

7. Repeat the previous three steps to add all the choices that should be presented to the user.



8. To change the position of a item in the list, select the item, and click the ‘Up’ or ‘Down’ arrow button.

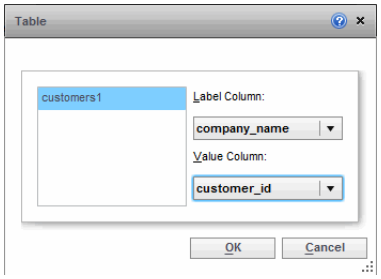
9. Click ‘OK’ to exit the ‘List’ dialog box.

6.2.3 Creating a Query-Based Variable List

To populate the Variable selection list with data obtained from a query (table), follow these steps:

1. In the ‘Selection List’ panel of the ‘Variable Properties’ dialog box, select the ‘Query’ option.

2. Click the ‘Select’ button to open the ‘Table’ dialog box.



3. From the list box on the left side, select the table that contains the data you wish to use.

4. From the ‘Value Column’ menu, select the column that contains the data to be used in the filtering condition. Each row of the ‘Value Column’ represents a possible value of the Variable.

5. In the ‘Label Column’ menu, select the column that contains the labels to be displayed to the user.

   Each row of the ‘Label Column’ should contain the label for the corresponding row of the ‘Value Column’. These labels are used to designate the available options in the ‘Enter Parameters’ dialog box that will be displayed to the user. When the user selects a particular option, the value corresponding to the selected label is applied to any filter conditions that use the Variable

6. Click ‘OK’ to exit the ‘Table’ dialog box.

7 Applications

This section describes high-level procedures used to approach certain tasks using the Data Worksheet.

These examples describe the general steps needed to apply Worksheet tools to an application. The example Worksheets can be found in the 'Global Worksheet' > 'Tutorial' folder. For specific details on how to carry out individual constituent tasks, e.g., grouping, concatenating tables, etc., please refer to those specific chapters in this guide.

7.1 User Conference Mailing

Compiling a custom mailing list for an upcoming conference is a necessary, but laborious, task. The functions provided by the Data Worksheet simplify this task. Say you are hosting a conference for select customers and you would like to send them printed invitations. Seating is limited, so sending an invitation to every customer would not be cost effective. Therefore, to narrow down the list of the customers to whom you will send the invitation, you can use their order history. Suppose you decide to send an invitation only to the following customers:

- The top 3 customers this year
- The customers who have purchased at least one of our products in the last four weeks

In this example, you will make use of the following components and methods:

- Advanced conditions (see *Defining Advanced Conditions*)
- Concatenated table options (see *Concatenating Tables*)
- Group and aggregate options (see *Grouping*)
- A table value in the filter conditions (see *Filtering*)

The completed example is called 'UserConferenceMailing', and can be found in the 'Global Worksheet' > 'Tutorial' folder.

Example: User Conference Mailing

Assume you have two tables: A table of order information for each customer, called 'OrderInfo', and a table of contact information for each customer, called 'ContactInfo'.

To build the desired invitation list, follow these steps:

1. Generate a list of all customers who have purchased at least one of our products in the last four weeks.

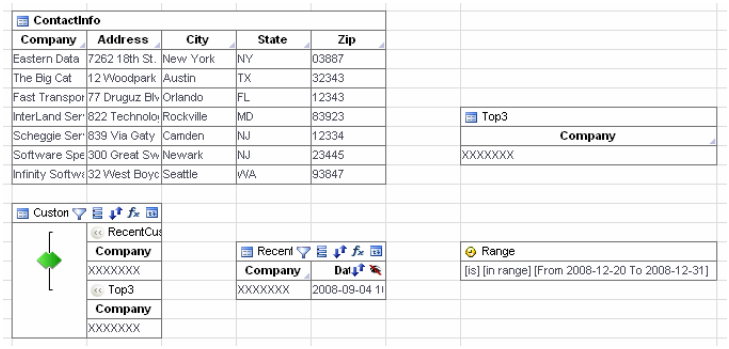
To do this, apply an ‘in range’ filter condition on the ‘OrderInfo’ table that restricts the data to ‘Last 4 Weeks’. Name this new table ‘RecentCustomers’. The ‘RecentCustomers’ table will to have two columns, ‘Customer Company’ and ‘Order Date’. You require the ‘Order Date’ field to specify the date condition, but the column can then be hidden.

2. Generate a list of the top 3 customers for this year. To do this, apply a ranking condition to the ‘OrderInfo’ table so that it only lists the top 3 customers based on the sum of the ‘Total’ sales for each customer.

3. Once you have both of these tables, you can unite them into a single table. Create a Composition table using the ‘union’ set operation. This requires that both base tables have the same number of columns, so hide all the columns of both tables except for the ‘Company Name’ columns. Name this new Concatenated table ‘CustomerList’. This table contains the list of Company names to whom you wish to send invitations.

4. You now need to retrieve the corresponding contact information for these companies. Specify a subquery-based filtering condition on the ‘Company’ field of the ‘ContactInfo’ table, so that a record is only displayed if the company is “one of” the companies in the ‘CustomerList’ table.

Figure 1. The ‘User Conference Mailing’ example



¢

7.2 What-If Analysis

What-if analysis is the exploration of cause and effect relationships in data. This is one of the greatest strengths of the Data Worksheet, making use of the following features:

- Embedded tables (see *Creating an Embedded Table*)

- Cell references in formulas (see Appendix B:*Accessing Table Cells in Script*)
- Dynamic table updating

Suppose that all of our big customers have been requesting discounts, and you therefore decide to give a 5% discount on all large orders. Since you need to recover the lost revenue incurred by this discount, you will simultaneously need to increase the sales total on all smaller orders by 5%.

To implement this discount plan, you need to decide where to place the boundary between “large order” and “small order” so as to maintain the company’s total revenue. This ‘price break’ value can be discovered through what-if analysis.

In the example below, you will make use of the following components and methods:

- Embedded Table (see *Creating an Embedded Table*)
- Joined Table (see *Joining Tables*)
- Concatenated Table (see *Concatenating Tables*)
- Group and aggregate options (see *Grouping*)
- Expression fields (see *Creating an Expression/Formula Column*)

The completed example is called ‘PriceBreakWhatIf’, and can be found in the ‘Global Worksheet’ > ‘Tutorial’ folder.

Example:
What-If Analysis

Assume you have a Worksheet table called ‘Revenue’. The ‘Revenue’ table includes the fields ‘Order Num’ and ‘Product Total’, listing the total value of each order. Follow the steps below to analyze this data and detect the proper ‘price break’ threshold for classifying orders as “large” or “small”:

1. Create a new Embedded Table with one data cell, and name this table ‘PriceBreak’. It will contain the ‘price break’ value that you experiment with. In the table cell, enter 40000 as the first guess for the price break value. This indicates that all orders with a total of $40,000 qualify as “large orders”, and all orders with a total equal to or less than $40,000 qualify as “small orders”.

Next, you will calculate the total sales of all of the large or ‘High End’ orders, and the total sales of all the small or ‘Low End’ orders, both before and after the discounts/increases are taken into account. You do this in order to compare the sales revenue before and after the discount model is put into effect.

2. In the ‘Revenue’ table, group the data by ‘Order Num’ and summarize by ‘Product Total’ so that you have the total sales amount for each order.

The data in the ‘Revenue’ table can now be split into two separate tables, ‘HighEnd’ and ‘LowEnd’.

3. To obtain the ‘HighEnd’ table, specify a join between the ‘Product Total’ column and the ‘PriceBreak’ table, and set the join type to:

   ```
   Revenue.Total >= PriceBreak.PriceBreak
   ```

4. To obtain the ‘LowEnd’ table, specify a join between ‘Product Total’ and ‘PriceBreak’ and set the join type to:

   ```
   Revenue.Total < PriceBreak.PriceBreak
   ```

5. For the ‘HighEnd’ table, create an expression field named ‘Discount’ to calculate the new order total with the 5% discount applied. The SQL formula for the ‘Discount’ expression field should be set to

   ```
   field['Total']*0.95
   ```

6. For the ‘LowEnd’ table, create an expression field named ‘Increase’ to calculate the new order total with the 5% increase applied. The SQL formula for the ‘Increase’ expression field should be set to

   ```
   field['Total']*1.05
   ```

Now that you have the new sales total values for the large and small orders, you can analyze the benefits of the current ‘price break’ value by comparing the difference between the original sales total and the new sales total.

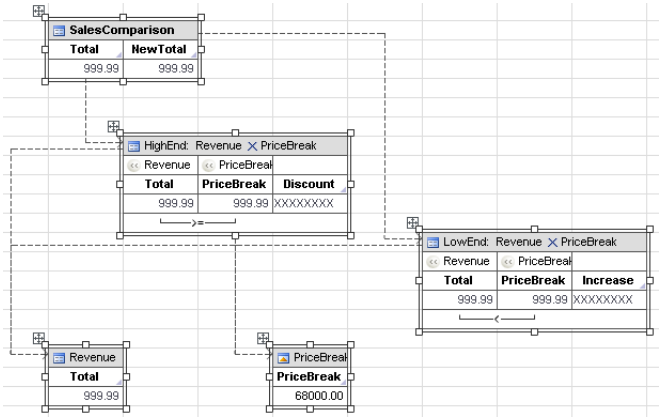
7. Create a union of the ‘HighEnd’ table and the ‘LowEnd’ table, and name this Concatenated Table ‘SalesComparison’. Name the right column ‘NewTotal’, because it contains the totals with discounts and increases applied.

8. Apply summarization on the ‘Total’ and ‘NewTotal’ fields of the ‘SalesComparison’ table to produce the final total revenue numbers.

Now every time you change the ‘price break’ value in the ‘PriceBreak’ embedded table, the values in the ‘SalesComparison’ table are automatically recalculated. After experimenting with a few numbers, you

learn that $50,000 is the optimal ‘price break’ for applying the new discount model without affecting the total company revenue.

Figure 2. Price break “What-If’ exploration



In addition to performing what-if experimentation by entering values manually into an embedded table, you can also adjust table values graphically by using a Viewsheet input element (slider, spinner, etc.). For more information about the embedded table interface to Viewsheets, see *Using Input Components* in *Advanced Dashboard Design*.

¢

APPENDIX A: Toolbar Buttons

Table 1. Toolbar buttons

BUTTON	FUNCTION
	Create a new Data Worksheet.
	Save the current Data Worksheet.
	Save the current Data Worksheet with a different name and possibly in a different scope.
	Set options relating to the entire Worksheet, such as table size limit and availability of data to reports.
	Preview the table that is set as Primary in the current Data Worksheet.
	Undo the changes.
	Reapply changes that were undone.
	Automatically lay out Worksheet assets with dependent elements immediately above their required assets.
	Prompt for all input parameters.
	Insert a new object into the Worksheet: Embedded Table, Variable, or Grouping.
	Apply a concatenation to two selected tables: Union, Intersection, or Minus.
	Apply a join to two selected tables: Merge Join, Cross Join, or Inner Join.
	Close the Visual Composer window.

APPENDIX B: Accessing Table Cells in Script

The power of a Worksheet comes from the ability to connect multiple assets together using drag and drop methods. However, you also provide a reference-based model for communication between assemblies, and this allows you to use arbitrary table values within expressions. (See *Creating an Expression/Formula Column* for more information about expressions.) This section describes this simple model for referencing table columns and cells.

B.1 Cell on the Current Row

Formulas run in the scope of the current row. Each cell value on the current row can be accessed by indexing into the 'field' array using the appropriate column name or column index. The syntax is as follows:

```
field['column name']
e.g.,
field['quantity']

field[column idx]
e.g.,
field[3]
```

For example, if 'quantity' and 'price' are two columns in the current table, a formula column in the same table can use the expression

```
field['quantity'] * field['price']
```

to produce the product "quantity*price" for every row of the 'quantity' and 'price' columns. To access the cell values of a different table, see Appendix B.4, *Cell of Any Table*.

B.2 Cell on a Previous Row

You can reference the values of preceding rows by their relative offset from the current row. This feature is useful when creating differential formulas. The syntax is as follows:

```
field[relative position]['column name']

e.g.,
field[-1]['Total Sales']  // previous row, in 'Total Sales'
field[-3][3]              // three rows above, in column 3
```

To reference a cell by its absolute position, see *Cell of Any Table*.

B.3 Column of Any Table

A column with name 'columnheader' in a Data Table with name 'Tablename' can be referenced as follows:

```
Tablename[columnheader]
```

B.4 Cell of Any Table

There are several ways to reference a cell in a table. To simply reference a cell in the current row of the current table, see Appendix B.1, *Cell on the Current Row*.

Referencing by Row Index

To reference an arbitrary cell in any table, you can specify the row location with a numeric index:

```
worksheet['Table Name']['columnheader'][row_ix]
```

If the table has no spaces in its name, you can use the simpler syntax below:

```
Tablename['columnheader'][row_ix]
```

For example, the reference

```
Customers['CompanyName'][5]
```

specifies the value in the 5th row (using 0-based indexing) of the 'CompanyName' column in the 'Customers' table.

Referencing by Value Lookup

Since most Data Tables are dynamic, referencing a cell by row index (see above) should only be used when a table contains a single row, or when rows are ranked. It is generally more useful to reference a cell by lookup using the table's primary key, as shown here:

```
Tablename["columnheader? primary_key == key_value"]
```

The "?" should be read as "where," so the above expression can be read as follows: "Look in the table named 'Tablename', and return the value in the column named 'columnheader' where the primary key is equal to 'key_value'."

For example, if a table named 'Customers' has 'customer_id' as its primary key, you can reference the cell in the 'CompanyName' column having key 'customer_id' by using the following expression:

```
Customers["CompanyName? customer_id == 123"]
```

Instead of using a fixed key value such as "123" for comparison, you can also use the field value of the current row in the current table. (See Appendix B.1, *Cell on the Current Row*, for the basic *field* syntax.) For example, consider a table named 'Customers', which contains columns named 'CompanyName' and 'customer_id' (the primary key). You can access values in the 'CompanyName' column as follows:

```
Customers["CompanyName? customer_id == field.local_id"]
```

This formula says: “Look in the ‘Customers’ table, and return the value in the column ‘CompanyName’ where the ‘customer_id’ key is equal to the value of the current cell in the ‘local_id’ column.”

Example:
Reference Cell

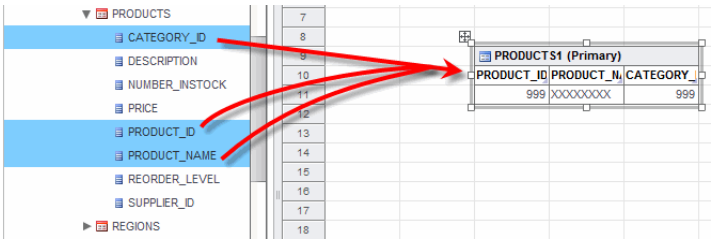
This example illustrates how you can reference the cells of another table from within a formula column. You will use the tables in the ‘TABLE’ > ‘SA’ folder of the ‘Orders’ data source to create the table below, which has columns for product ID, product name, category ID, and category name.

products

product_id	product_name	cat_id	cat_name
1	Xconnect Server	3	Business
2	InsideView	3	Business
3	Info Folder	6	Office Tools
4	Green Planet	4	Personal

1. Drag the following fields from the ‘TABLE’ > ‘SA’ > ‘PRODUCTS’ table to an empty cell on the Worksheet. (Option-click the field names to select them all, and then drag them together.)

   ```
   PRODUCT_ID
   PRODUCT_NAME
   CATEGORY_ID
   ```



   This creates a new table named ‘PRODUCTS1’.

2. Rename the ‘CATEGORY_ID’ column as ‘CAT_ID’. This will help to distinguish it from the table you will add next.

3. Drag the ‘TABLE’ > ‘SA’ > ‘CATEGORIES’ table to the Worksheet. This creates a new table named ‘CATEGORIES1’.

CATEGORIES1

CATEGORY_	CATEGORY_
999	XXXXXXXX

You now want to add a column to the ‘PRODUCTS1’ table that will contain the category name corresponding to each entry in the ‘CAT_ID’ column. The best way to do this is by *joining* the tables (see the *Joining Tables* section), but to practice using cell references, you will create this new column as a formula column.

4. Click the '*fx*' button in the 'PRODUCTS1' table title row. This opens the 'Expression' dialog box. (See *Creating an Expression/ Formula Column* for more information.)

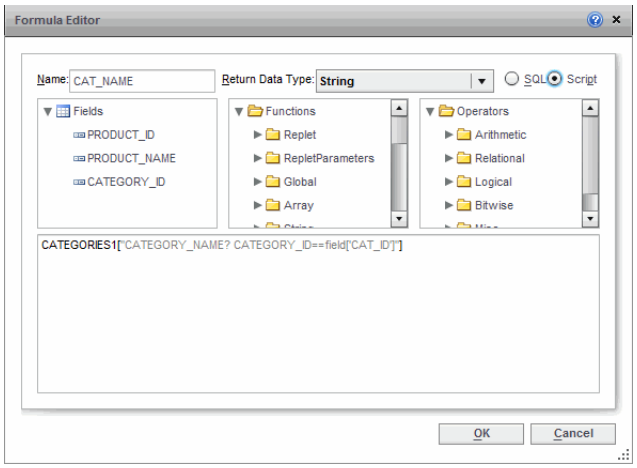
5. Enter 'CAT_NAME' as the name of the new expression column, and click 'OK'. This opens the Formula Editor.

6. In the Formula Editor, enter the following formula:

   ```
   CATEGORIES1["CATEGORY_NAME? CATEGORY_ID==field['CAT_ID']"]
   ```

   This formula says: "Look in the 'CATEGORIES1' table, and return the value in the column 'CATEGORY_NAME' where the 'CATEGORY_ID' key is equal to the value of the current cell in the 'CAT_ID' column."

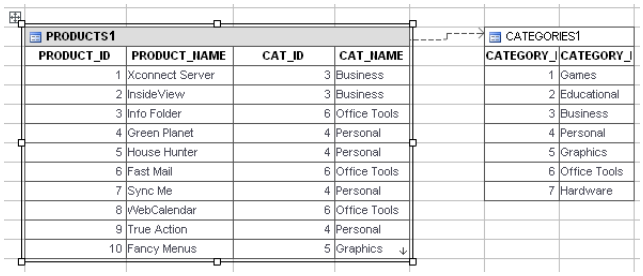
7. Uncheck the 'SQL' option at the top-right of the Editor.



8. Click 'OK' to close the Formula Editor. The two tables are now connected by a graphical link to indicate that the 'PRODUCTS1' table references the 'CATEGORIES1' table.

9. For each table, press the 'Change View' button in the table title bar, and select 'Live Preview' from the menu. Expand the table borders as needed to view the results.

   As desired, the formula column in the 'PRODUCTS1' table now displays the category name corresponding to each entry in the 'CAT_ID' column.



PRODUCTS1

PRODUCT_ID	PRODUCT_NAME	CAT_ID	CAT_NAME
1	Xconnect Server	3	Business
2	InsideView	3	Business
3	Info Folder	6	Office Tools
4	Green Planet	4	Personal
5	House Hunter	4	Personal
6	Fast Mail	6	Office Tools
7	Sync Me	4	Personal
8	WebCalendar	6	Office Tools
9	True Action	4	Personal
10	Fancy Menus	5	Graphics

CATEGORIES1

CATEGORY_I	CATEGORY_I
1	Games
2	Educational
3	Business
4	Personal
5	Graphics
6	Office Tools
7	Hardware

¢

APPENDIX C: Built-in Ranges and Groupings

This appendix lists various predefined ranges and groups.

C.1 Built-in Date Ranges

The following built-in date ranges are available in filtering conditions for Data tables. See the *Filtering* section for information about using these predefined ranges.

Last year	Last month last year	Last July
This year	This month last year	Last August
Last quarter	This January	Last September
This quarter	This February	Last October
Last quarter last year	This March	Last November
This quarter last year	This April	Last December
1st Quarter this year	This May	Last week
2nd Quarter this year	This June	This week
3rd Quarter this year	This July	Week before last week
4th Quarter this year	This August	Last 7 days
1st Quarter last year	This September	Last 8-14 days
2nd Quarter last year	This October	Last 4 weeks
3rd Quarter last year	This November	Last 5-8 weeks
4th Quarter last year	This December	Last 30 days
1st half of this year	Last January	Last 31-60 days
2nd half of this year	Last February	Year to date last year
1st half of last year	Last March	Month to date last month
2nd half of last year	Last April	Quarter to date last quarter
Last month	Last May	Month to date last year
This month	Last June	Quarter to date last year

The explanation of the last five ranges is as follows:

- *Year to date last year*: The same date range provided by the 'Year to date' option (i.e., beginning of year to present date), but offset to the previous year. For example, if today is Nov. 8, 2011, the range selected by 'Year to date last year' is Jan. 1, 2010–Nov. 8, 2010.
- *Month to date last month*: The same date range provided by the 'Month to date' option (i.e., beginning of month to present date), but offset to the previous month. For example, if today is Nov. 8, 2011, the range selected by 'Month to date last month' is Oct. 1, 2011–Oct. 8, 2011.
- *Quarter to date last quarter*: The same date range provided by the 'Quarter to date' option (i.e., beginning of quarter to present date), but

offset to the previous quarter. For example, if today is Nov. 8, 2011, the range selected by ‘Quarter to date last quarter’ is July. 1, 2011–Aug. 8, 2011.

- *Month to date last year*: The same date range provided by the ‘Month to date’ option (i.e., beginning of month to present date), but offset to the previous year. For example, if today is Nov. 8, 2011, the range selected by ‘Month to date last year’ is Nov. 1, 2010–Nov. 8, 2010.
- *Quarter to date last year*: The same date range provided by the ‘Quarter to date’ option (i.e., beginning of quarter to present date), but offset to the previous year. For example, if today is Nov. 8, 2011, the range selected by ‘Quarter to date last quarter’ is Oct. 1, 2010–Nov. 8, 2010.

C.2 Built-in Date Groupings

The following built-in date groupings are available for columns with a Date format.

Table 2. Built-in date grouping

DATE GROUP	VALUE IN THE GROUP COLUMN
Year	Date grouped at the year level
Quarter	Date grouped at the quarter level
Month	Date grouped at the month level
Week	Date grouped at the week level
Day	Date grouped at the day level
Hour	Date grouped at the hour level
Minute	Date grouped at the minute level
Second	Date grouped at the second level
Quarter of Year	Ordinal number: 1st, 2nd, 3rd, 4th
Month of Year	Month name: Jan, Feb, etc.
Week of Year	Numbers 1-52
Day of Month	Numbers 1-31
Day of Week	Day of week name: Sun, Mon, etc.
Hour of Day	Numbers 0-23

See the *Grouping* section for information about using these predefined groupings. See *Creating a Named Grouping* for information about creating alternative groupings.

Advanced Dashboard Design

Visualization is a branch of analysis that turns data into visual insight. Traditional data visualization tools were designed for highly trained professionals, such as scientific researchers. These used complicated outputs and statistical models to enable these users to discover low-level patterns.

A Viewsheet is a business intelligence dashboard that brings the power of visualization to business executives and mainstream users. Because Viewsheets allow you to interact with data using familiar objects such as charts, sliders, and check-boxes, they are quick to learn and easy to use. They allow you to seamlessly integrate data warehouses and other BI data stores with operational data sources for a complete view of your organization.

Note: Viewsheets are available only in Style Intelligence and Style Scope.

Visualization using Viewsheets goes far beyond the ad hoc querying you might use to find answers to existing questions. With Viewsheets, you will be able to explore your data from every angle, discover new patterns, test new ideas, and envision new opportunities.

1 Visual Composer

Visual Composer is a web-based tool for creating Data Worksheets and dashboards (Viewsheets). You will use the Visual Composer to link graphical components of a Viewsheet to the data supplied by a Worksheet, query, data model, or database table.

Note: Visual Composer is available only in Style Intelligence and Style Scope.

This chapter discusses some basic aspects of the Visual Composer interface. For a quick start on creating dashboards, see *Creating a Dashboard*.

See Also

Advanced Data Mashup, for information on creating Worksheets and other assets.

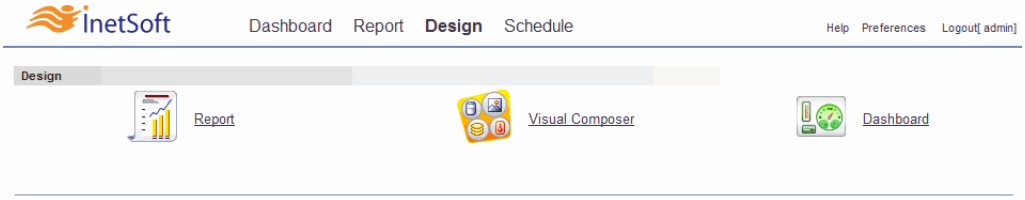
Advanced Data Modeling, for information on creating queries and data models.

1.1 Launching Visual Composer

To launch Visual Composer, follow these steps:

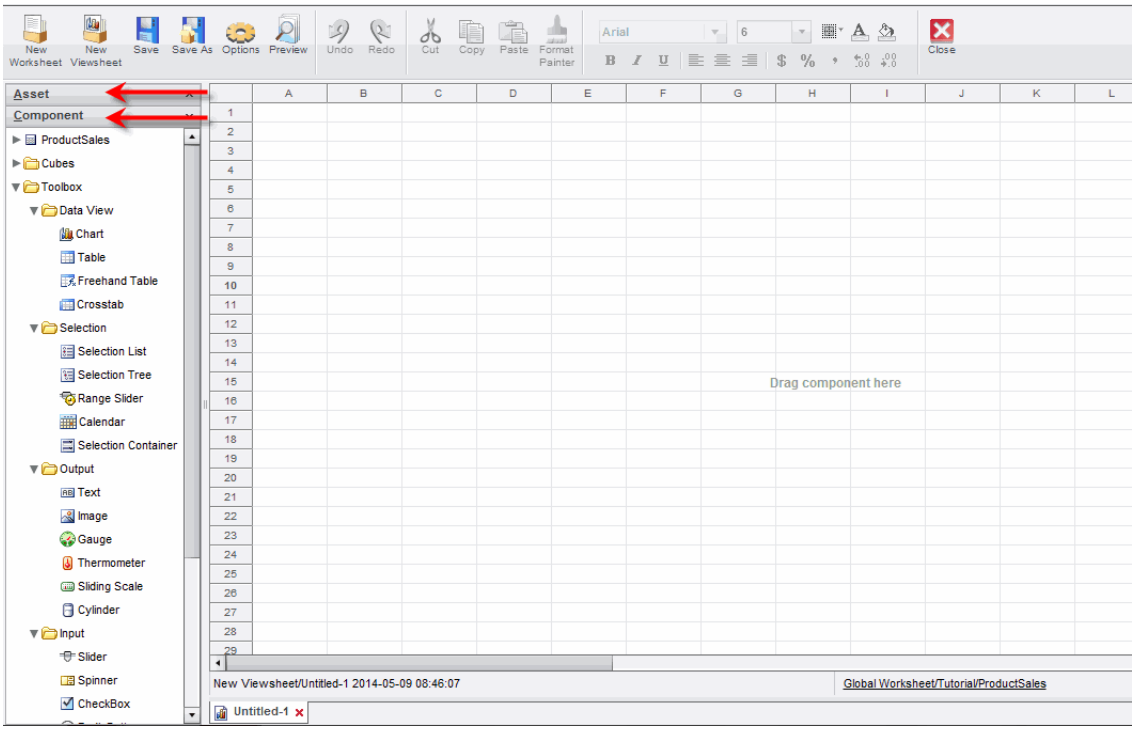
1. Open the User Portal within a web browser. (Your administrator can provide the correct web address.)
2. In the User Portal, select the **Design** tab.
3. Click the 'Visual Composer' link.





The pane on the left side of the Visual Composer contains two collapsible panels, **Asset** and **Component**. You can toggle between the two panels by clicking on their respective title bars.

- The **Asset** panel lists all Data Worksheets and Viewsheets.
- The **Component** panel lists all the components and data sources available to the Viewsheet.



1.2 Managing Viewsheets

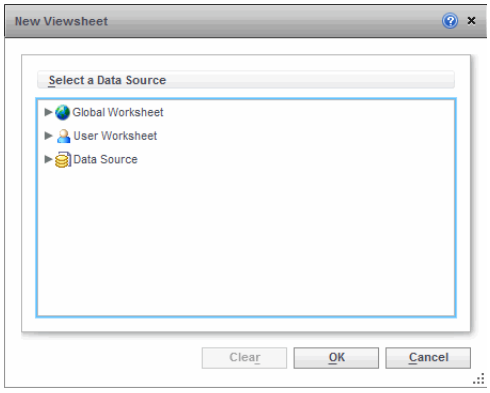
The following sections explain basic Viewsheet operations.

1.2.1 Creating a New Viewsheet

To create a new Viewsheet, follow these steps:



1. Click the 'New Viewsheet' button on the main toolbar. This opens the 'New Viewsheet' dialog box.



2. Select a data source for the Viewsheet:

a. Queries, data models, and database tables are listed under the ‘Query’ node. The Viewsheet will have access to all of the fields in the selected query, model, or table.

b. Data Worksheets are listed under the ‘Global Worksheet’ and ‘User Worksheet’ nodes. The Viewsheet will have access to all of the Data Blocks in the selected Worksheet.

You are not required to select a data source. If you wish to create the Viewsheet without an associated data source, do not make any selections here. (To clear an existing selection, press the ‘Clear’ button.) To bind the Viewsheet to a data source at a later time, see *Linking a Viewsheet to a Data Source*.

3. Click ‘OK’.

1.2.2 Opening an Existing Viewsheet

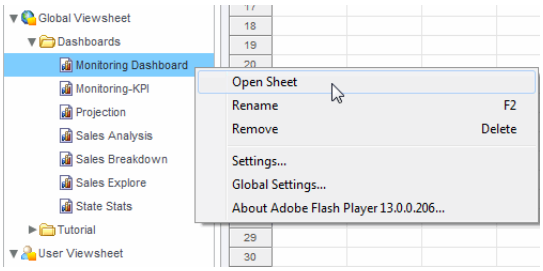
To open an existing Viewsheet, follow these steps:

Viewsheets under ‘Global Viewsheet’ are visible to all users of the Visual Composer. Viewsheets under ‘User Viewsheet’ are visible only to their creator.

1. Select the **Asset** panel of the Visual Composer. The asset tree contains two Viewsheet nodes labeled ‘Global Viewsheet’ and ‘User Viewsheet’.

2. Expand the nodes to see the list of Viewsheets.

3. Right-click the Viewsheet you wish to open, and select ‘Open Sheet’ from the context menu.



Alternatively, you can double-click the Viewsheet you wish to open.

If you press the ‘Cancel’ button in the Viewsheet loading dialog box, the Viewsheet is automatically displayed in meta-data mode. See *Suppressing Query Execution in Design View* for more information about meta-data mode.

See Also

Linking a Viewsheet to a Data Source, to link an existing Viewsheet to data source.

Controlling Access to an Asset, in *Advanced Data Mashup*, for information on Worksheet scope.

1.2.3 Saving a Viewsheet

This section illustrates how to save a Viewsheet, and explains the different storage scopes available.

Saving a New Version of an Existing Viewsheet



To save a new version of an existing Viewsheet, click the ‘Save’ button in the toolbar, or press Ctrl-S on the keyboard. This replaces the older version with the current version, retaining the same name.

```
Note: You must possess write permission for the Viewsheet or a
folder in order to save changes. The 'Save As' option is not
available in Style Scope Free Edition.
```



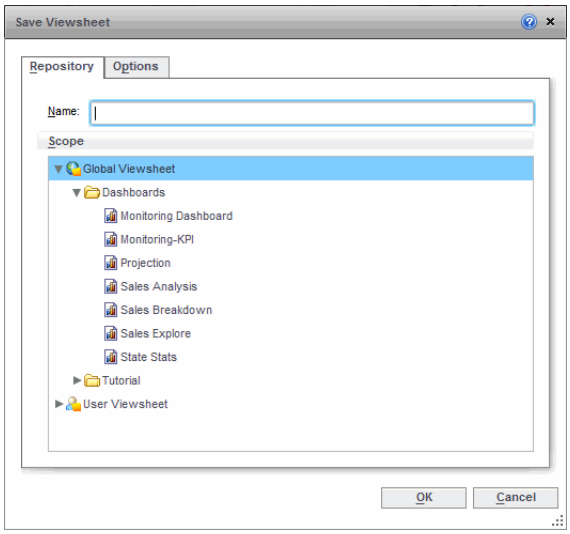
To save the new version of the Viewsheet under a different name, click the ‘Save As’ button. This opens the ‘Save Viewsheet’ dialog box. Follow the instructions in *Saving a New Viewsheet*.

Saving a New Viewsheet

To save a new Viewsheet, follow the steps below:



1. Click the ‘Save’ button in the toolbar, or press Ctrl-S on the keyboard. This opens the ‘Save Viewsheet’ dialog box.



2. Under the **Repository** tab, enter a name for the Viewsheet in the ‘Name’ field.

3. Select the appropriate scope for the saved Viewsheet (‘Global Viewsheet’, ‘User Viewsheet’, or a sub-folder).

```
Note: You must possess write permission for the target scope (sub-
folder).
```

4. Under the **Options** tab, select ‘List on Portal Tree’ to make the Viewsheet available to users in the User Portal.

5. Click ‘OK’ to save the Viewsheet.

The Viewsheet is saved to the Asset Repository, and appears in the Asset Repository tree under the specified scope. Saved Viewsheets are stored in a file called *asset.dat* within the SREE Home directory.

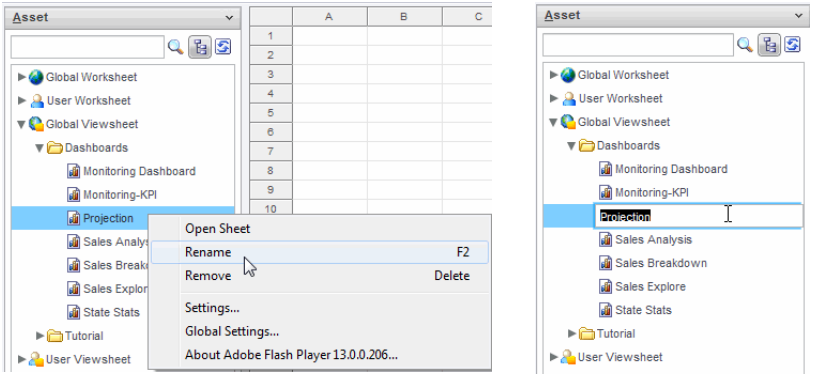
See Also
Deploying a Viewsheet, for information on Portal access to Viewsheets.
Setting Viewsheet Options, for general Viewsheet options.
Controlling Access to a Viewsheet, for information on setting Viewsheet scope.

1.2.4 Renaming a Viewsheet

To rename a Viewsheet, follow these steps:

1. Select the Viewsheet in the Asset Repository.

2. Click the Viewsheet name a second time, or right-click and select ‘Rename’. This activates the edit mode.



3. Enter the new name, and press Enter.

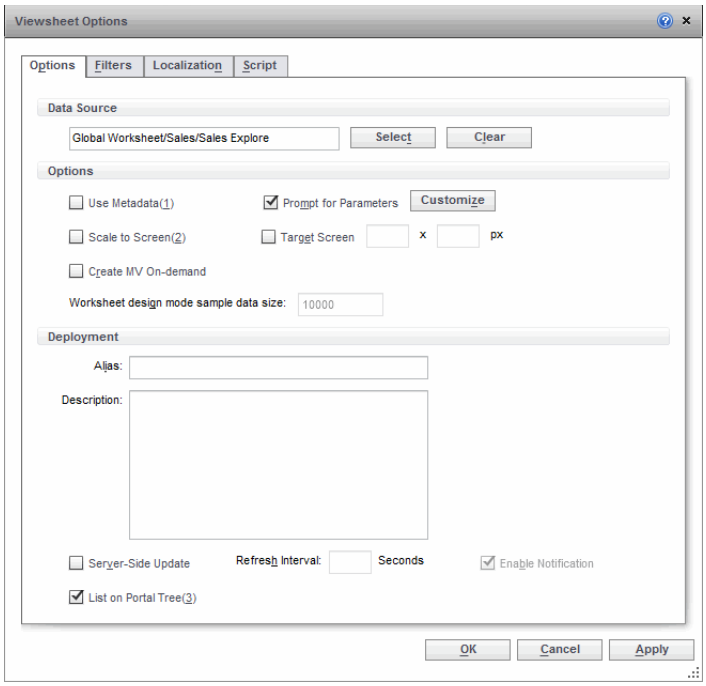
1.2.5 Linking a Viewsheet to a Data Source

To bind a Viewsheet to a new data source, follow the steps below:

1. Open the Viewsheet in the Visual Composer.

2. Click the ‘Options’ button in the Visual Composer toolbar. This opens the ‘Viewsheet Options’ dialog box.

3. In the ‘Data Source’ region, click the ‘Select’ button.



4. Select a data source for the Viewsheet:

   a. Queries, data models, and database tables are listed under the ‘Query’ node. The Viewsheet will have access to all of the fields in the selected query, model, or table.

```
Note: In general, if a Viewsheet contains controls that are already
bound to a Worksheet, those controls will stop functioning when the
Viewsheet is re-bound to a new Worksheet. However, the controls
will remain functional if the Data Blocks and columns in the new
Worksheet are named the same as the Data Blocks and columns in the
old Worksheet.
```

   b. Data Worksheets are listed under the ‘Global Worksheet’ and ‘User Worksheet’ nodes. The Viewsheet will have access to all of the Data Blocks in the selected Worksheet.

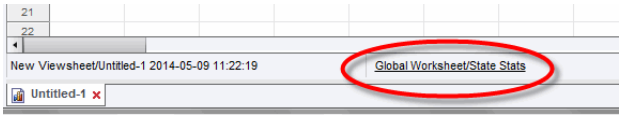
5. Click ‘OK’.

See Also

Controlling Access to an Asset, in *Advanced Data Mashup*, for information on the Worksheet scopes.

1.2.6 Opening a Linked Worksheet

To open the Worksheet linked to a Viewsheet, click the Worksheet link at the bottom of the Viewsheet.



1.2.7 Controlling Access to a Viewsheet

You can store Viewsheets under two different scopes. The scope determines which users can access the Viewsheet in the Portal, Visual Composer, and Dashboards.

The following two scopes are available for Viewsheets:

- **Global scope**: All of the users in the system have potential access to the Viewsheet. The administrator assigns individual user permissions.
- **User scope**: Only the user who creates the Viewsheet can access it.

For Global scope only, an administrator can define subdirectories to organize Viewsheets into different domains, for example:

– Shared Viewsheets
– Sales domain
– Marketing domain
– R&D domain
– Support domain
– Financial domain

Each domain can be given different access rights.

See Also
Changing Viewsheet Scope, for information on toggling Global and Local scope.
Creating Folders, in *Advanced Server Administration*, to define sub-folders.
Repository Permissions, in *Advanced Server Administration*, to assign permissions.

1.2.8 Changing Viewsheet Scope

To change the scope of a Viewsheet in the Asset Repository, drag the Viewsheet to the branch of the Repository tree corresponding to the desired scope. The following rules apply:

- You must possess write permission to the target scope.
- If the Viewsheet is linked to a Worksheet that has User scope, the Viewsheet cannot be moved to Global scope.

See Also

Controlling Access to an Asset, in *Advanced Data Mashup*, for information on the Worksheet scopes.

1.2.9 Limiting Data Retrieved by a Viewsheet

It is common for administrators to limit data retrieved by user queries, in order to avoid overtaxing database resources. Administrators can limit the number of records retrieved in design mode, preview mode, and at runtime (in the Portal). The design mode limit can also be set in the Worksheet. See the related links for more information.

In cases where a Worksheet Data Block is slow or returns an empty result set, you can apply formatting features (e.g., highlights) by *Suppressing Query Execution in Design View*.

See Also

Web Properties and Safeguards, in *Advanced Server Administration*
Limiting Query Size and Execution Time, in *Advanced Server Administration*, for global safeguards.
Setting Global Worksheet Options, in *Advanced Data Mashup*, to limit Viewsheet data in design view.
Limiting the Number of Rows in a Table, in *Advanced Data Mashup*, for other Worksheet-level safeguards.

1.2.10 Suppressing Query Execution in Design View

Note: If you press the 'Cancel' button in the Viewsheet loading dialog box, the Viewsheet is automatically displayed in meta-data mode.

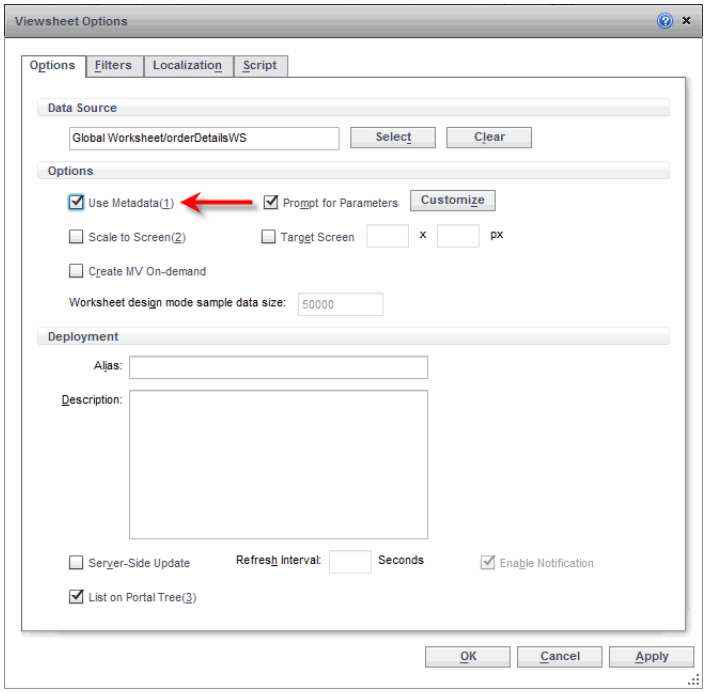
The design view for a Viewsheet is intended to simulate the functional Viewsheet as closely as possible, including all data representation and interactivity. However, when a Viewsheet relies on large or complex queries, retrieval of data from the database at design time can introduce unnecessary delays into the design process. In such cases, it is convenient to suppress the execution of queries in the Viewsheet design view and instead display only meta-data.

To manually suppress execution of queries in the design view, follow the steps below:

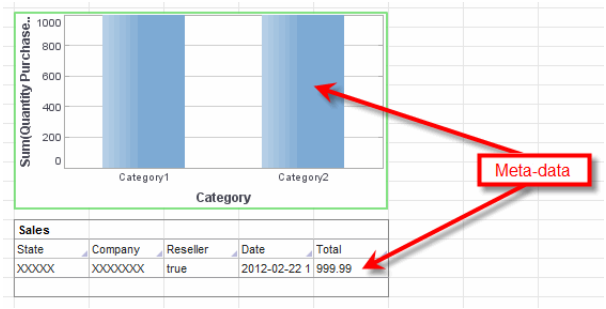


1. Click the 'Options' button in the Visual Composer toolbar. This opens the 'Viewsheet Options' dialog box.

2. Select the 'Use Metadata' option, and click 'OK'.



With 'Use Metadata' enabled, data-view and selection components in the design view display query *meta-data* rather than the query result set.

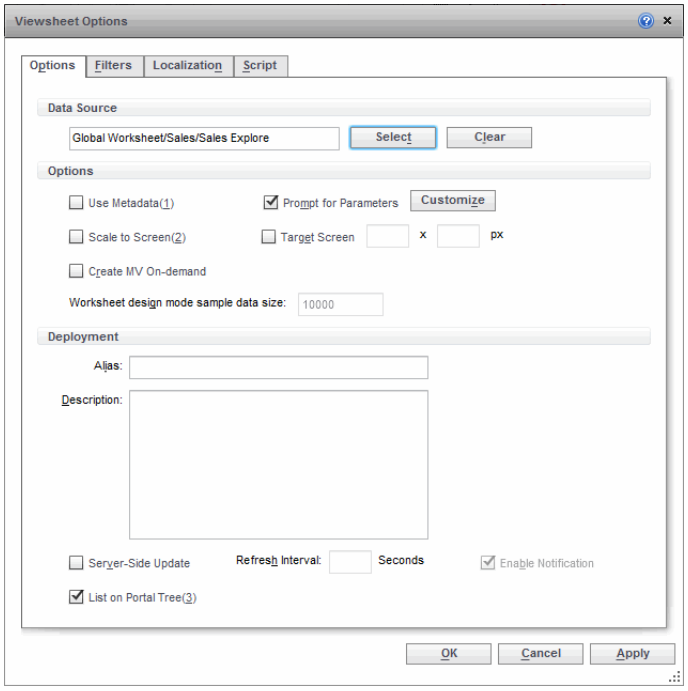


1.2.11 Suppressing Parameter Prompting

By default, a Viewsheet will prompt the user for all parameters in the linked data source (query, Data Worksheet, etc.) and have not been assigned a value. In some cases, you may want to suppress the prompting for some or all of these parameters.

To suppress prompting for all parameters, follow the steps below:

1. Press the 'Options' button in the toolbar to open the 'Viewsheet Options' dialog box.



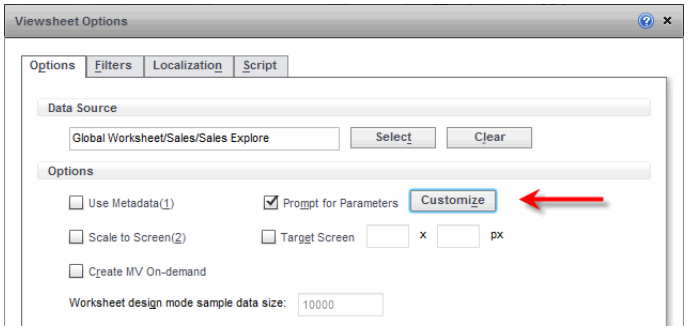
2. Disable the ‘Prompt for parameters’ option.

3. Press ‘OK’ to close the dialog box.

To suppress prompting for a subset of parameters, follow the steps below:

1. Press the ‘Options’ button in the toolbar to open the ‘Viewsheet Options’ dialog box (as above).

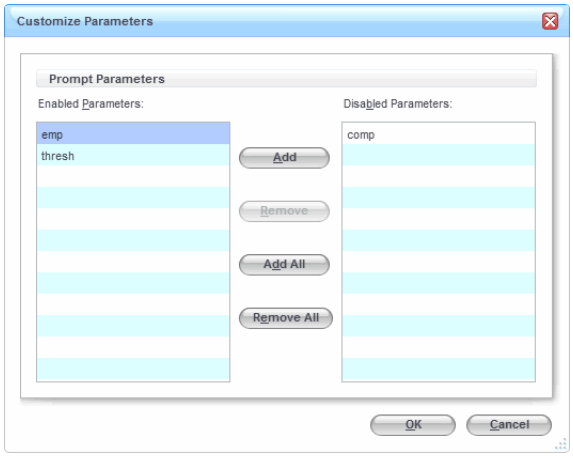
2. Press the ‘Customize’ button next to the ‘Prompt for parameters’ option.



This open the ‘Customize Parameters’ dialog box.

3. To disable one or more parameters, select the parameters you wish to disable in the ‘Enabled Parameters’ panel, and press the ‘Add’

button. This moves the selected parameters to the ‘Disabled Parameters’ panel.



To re-enable one or more disabled parameters, select the parameters you wish to enable in the ‘Disabled Parameters’ panel, and press the ‘Remove’ button. This moves the selected parameters to the ‘Enabled Parameters’ panel.

4. Press ‘OK’ to close the ‘Customize Parameters’ dialog box.

5. Press ‘OK’ to close the ‘Viewsheet Options’ dialog box.

See Also

Setting Viewsheet Options, for more information about the ‘Viewsheet Options’ dialog box.

1.3 Configuring Viewsheet Appearance

This section discusses several ways that you can modify the overall appearance of a Viewsheet. These include alterations you can make to the Viewsheet grid, and the use of tabbed controls and nested Viewsheets.

1.3.1 Viewsheet Toolbar

The table below describes the Visual Composer toolbar buttons available for editing Viewsheets

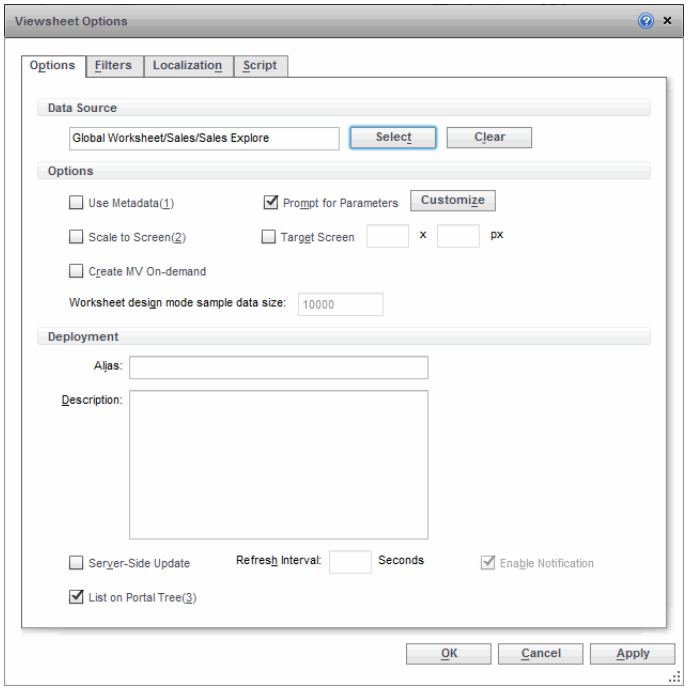
BUTTON	FUNCTION
	New Worksheet: Create a new Data Worksheet.
	New Viewsheet: Create a new Viewsheet based on a Worksheet, query, data model, or database table.
	Save: Save the changes to Viewsheet. (Ctrl-S)

	Save As: Save the Viewsheet under a new name or scope.
	Options: Settings for the Viewsheet as a whole.
	Preview: View the Viewsheet as the user does.
	Undo: Undo the last operation. (Ctrl-Z)
	Redo: Redo the last operation that was undone. (Ctrl-Y)
	Format Painter: Copy text formatting from one object to another.
	Currency Format: Apply a currency format.
	Percentage Format: Apply a percentage format.
	Comma Style: Apply a number format.
	Increase Decimal: Increase the number of decimal places.
	Decdecrease Decimal: Decrease the number of decimal places.
	Borders: Apply borders to selected objects.
	Font color: Select text color.
	Fill color: Select background color of objects
	Close: Close the Visual Composer

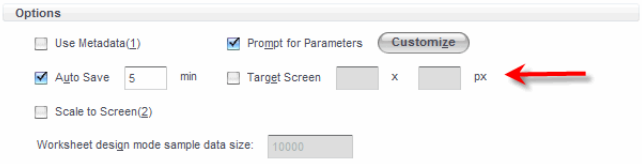
1.3.2 Tailoring the Viewsheet for Specific Screen Size

To lay out a Viewsheet to fit particular screen dimensions, use the ‘Target Screen’ option to display appropriate guidelines on the Viewsheet grid. Follow the steps below:

1. Press the ‘Options’ button in the toolbar to open the ‘Viewsheet Options’ dialog box.

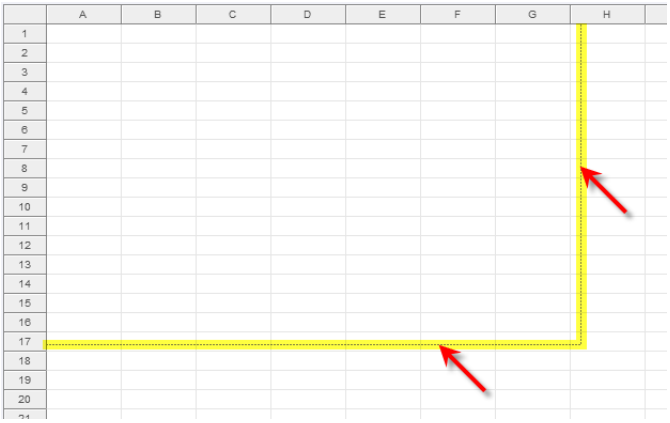


2. Enable the ‘Target Screen’ option, and enter the desired screen dimensions in pixels.



3. Press ‘OK’ to close the dialog box.

The Viewsheet grid now displays dashed border lines defining a region of the size that you specified.



By laying out the Viewsheet components within these border lines, you can ensure that the Viewsheet will fit within a screen of the specified dimensions.

See Also

Setting Viewsheet Options, for more information about the 'Viewsheet Options' dialog box.

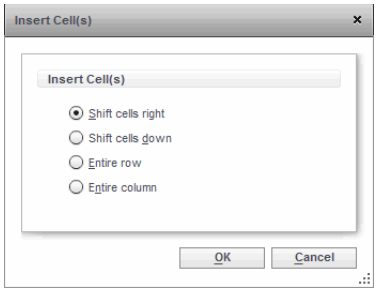
1.3.3 Modifying the Viewsheet Grid

It is possible to alter the Viewsheet grid by inserting or deleting cells, rows, and columns. You can also resize the grid rows and column, and make the grid invisible.

Inserting a Cell, Row, or Column into a Viewsheet

To insert a new cell, row, or column into the Viewsheet grid, follow these steps:

1. Right-click on the cell, row, or column where you want the insertion to take place, and select 'Insert Cell(s)' from the context menu. This opens the 'Insert Cell(s)' dialog box.



2. In the 'Insert Cell(s)' dialog box, select the desired insertion type:

   a. To insert a single blank cell at the current grid location, select 'Shift cells down' or 'Shift cells right'.

      This inserts the new cell, and displaces the currently selected cell (and all subsequent cells) downward or to the right, respectively.

   b. To insert a blank row or column at the current grid location, select 'Entire row' or 'Entire column'.

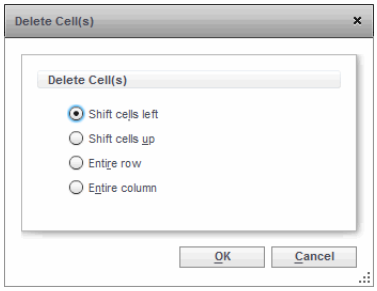
      This inserts the new row or column, and displaces the currently selected row or column (and subsequent rows or columns) downward or to the right, respectively.

Note: The position of a Viewsheet component will be shifted only if the new cell is inserted in the same row or column as the grid cell occupied by the component's top-left corner.

Deleting a Cell, Row, or Column from the Viewsheet

To delete a cell, row, or column from the Viewsheet grid, follow these steps:

1. Right-click the cell, row, or column you want do delete, and select ‘Delete Cell(s)’ from the context menu. This opens the ‘Delete Cell(s)’ dialog box.



2. In the ‘Delete Cell(s)’ dialog box, select the desired deletion type:

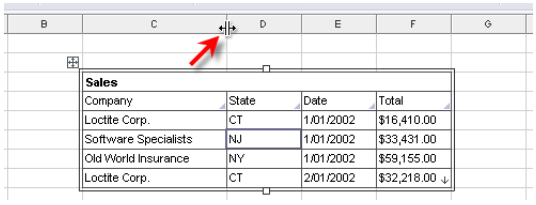
   a. To delete the single cell at the current grid location, select ‘Shift cells up’ or ‘Shift cells left’. This deletes the currently selected cell, and displaces the following cells upwards or to the left, respectively.

   b. To delete the currently selected column or row, select ‘Entire row’ or ‘Entire column’. This deletes the current row or column, and displaces the following rows or columns upward or to the left, respectively.

Resizing the Viewsheet Grid

To resize a row or column of the Viewsheet grid, follow these steps:

1. Move the mouse over a divider in the row or column header region. The pointer changes to a resize tool.



2. Drag the row or column divider to change the row height or column width.

See Also

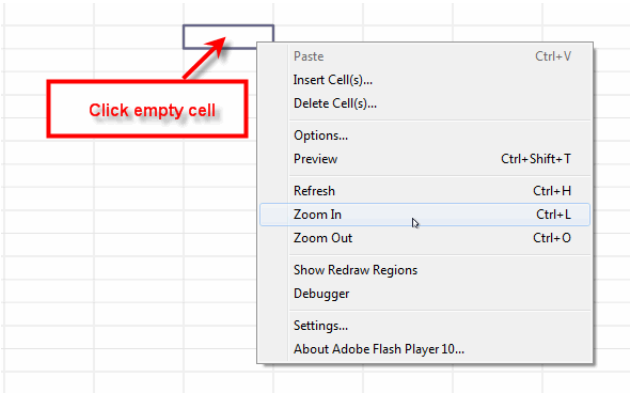
Resizing a Table Row, to adjust row size independently of the grid.

Resizing a Table Column, to adjust table cell size independently of the grid.

Zooming the Viewsheet Grid

The Viewsheet grid provides a “zoom” feature that uniformly scales the entire Viewsheet. This allows you to simultaneously reduce or enlarge all components on the Viewsheet while keeping their relative positions and sizes intact.

To zoom into or out of a Viewsheet, right-click in an *empty region* of the grid, and select ‘Zoom In’ or ‘Zoom Out’ from the context menu.



The Viewsheet remains fully functional even while it is in a zoomed state.

See Also

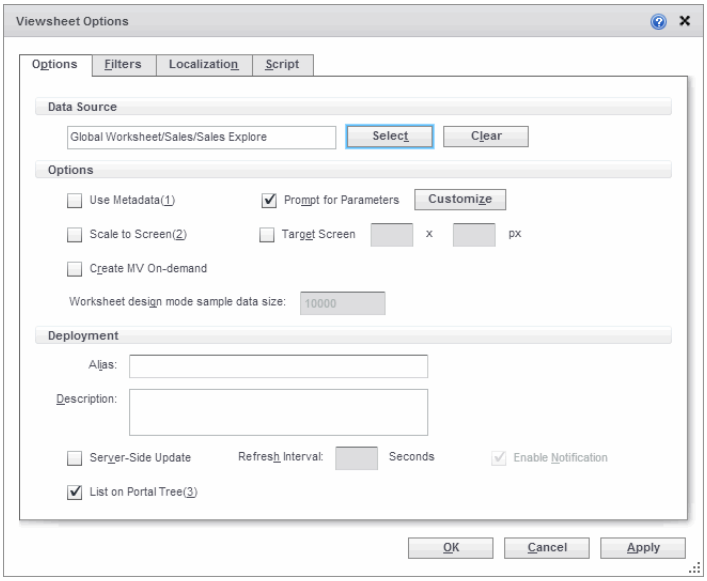
Resizing a Component, to adjust the size of an individual component.

1.3.4 Setting Viewsheet Options



You can set global options for the Viewsheet from the ‘Viewsheet Options’ dialog box. To open the ‘Viewsheet Options’ dialog box, do one of the following:

- Click the ‘Options’ button in the main toolbar.
- Right-click in an empty location on the Viewsheet grid, and select ‘Options’ from the context menu.



The **Options** tab of the 'Viewsheet Options' dialog box provides the following settings:

SETTING	DESCRIPTION
Data Source	The Worksheet, query, data model, or table to which the current Viewsheet is linked. Press 'Select' to choose a data source. Press 'Clear' to unbind the data source.
Use Metadata	Suppress the display of live data during Viewsheet design. See *Suppressing Query Execution in Design View*.
Prompt for parameters	Enables user prompting for any parameters defined in the linked data source. Disable 'Prompt for parameters' to completely suppress prompting. Press the 'Customize' button to selectively disable prompting for certain parameters. (If disabled parameters are used within filter conditions, the corresponding filter conditions are dropped.) See *Suppressing Parameter Prompting* for more information.
Target Screen	Displays a dashed border of the specified pixel dimensions on the Viewsheet grid to assist with Viewsheet sizing. See *Tailoring the Viewsheet for Specific Screen Size* for more information.
Scale to Screen	When enabled, scales the Viewsheet to utilize the entire width and height of the browser window so that all components are visible without scrolling.
Fit to Width	When enabled, scales the Viewsheet to the full width of the browser window. The Viewsheet height is scaled proportionally to the width, and may therefore exceed the height of the window. (A vertical scroll bar is provided in this case.)

SETTING	DESCRIPTION
Create MV On-demand	Specifies that a new materialized view should be automatically created on-the-fly if the Viewsheet does not have an existing materialized view (either because the materialized view was never populated with data or because the Viewsheet underwent modifications that rendered an existing materialized view unusable). The materialized view will supply data to the Viewsheet both at design time and at runtime.
Server-Side Update	Automatically refreshes Viewsheet data at the period specified by the 'Refresh Interval'. To customize the refresh behavior, see *Enabling Server-Side Update for a Dashboard* in *Advanced Server Administration*. If a materialized view has been created for the Viewsheet, server-side update is inactive. (See *Materialized Views* in *Advanced Server Administration* for more information on materialized views.) To suppress the "spinning wheel" loading graphic during automatic updates, deselect the 'Display Loading Indicator' property.
Design mode sample data size	Limits the number of records retrieved by a Viewsheet in design mode. This setting applies only to Viewsheets based on queries and data models. Viewsheets that are based on a Data Worksheet are governed by the corresponding 'Design mode sample data size' setting in the Worksheet, and the Viewsheet setting is disabled. See *Setting Global Worksheet Options* in *Advanced Data Mashup* for more information.
List on Portal Tree	Determines whether the current Viewsheet will be visible in the User Portal (subject to permissions set by administrator).
Alias	Specifies a designation to be used in place of the actual Viewsheet name. The alias replaces the Viewsheet name in all contexts except the following: The **Report** tab and the 'Export Assets' page in Enterprise Manager. These always display the actual Viewsheet name.
Description	A description to appear as a tooltip when the user hovers the mouse over the Viewsheet in the Portal tree.

The **Filters** tab of the 'Viewsheet Options' dialog box allows you to specify the manner in which selection elements in different Viewsheets should interact within the Portal. See *Sharing Filters* for more information.

The **Localization** tab allows you to identify Viewsheet elements that should be translated into one or more other languages based on user locale. See *Localization/Internationalization* for more information.

See Also

Linking a Viewsheet to a Data Source, to configure a Viewsheet data source.

Deploying a Viewsheet to the Repository, for information on displaying reports.

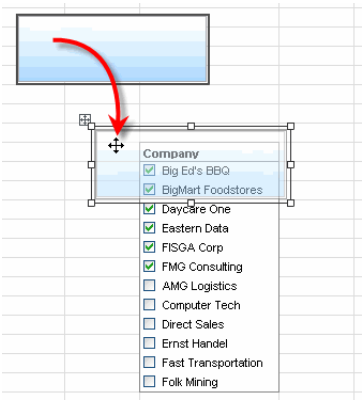
Using a Parameter in a Condition, in *Advanced Data Mashup*, for parameter usage.

1.3.5 Creating a Background

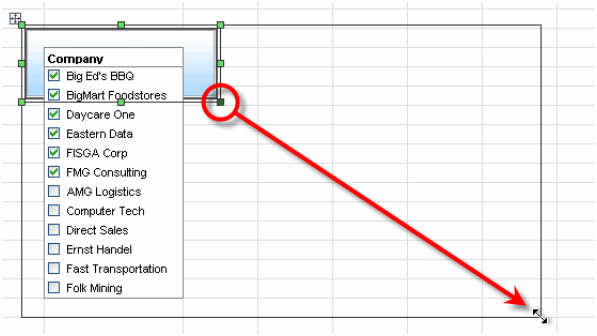
You can use an image or a shape component as a background for any portion of a Viewsheet. For example, you can place an image behind a set of components to provide the appearance of a frame or panel.

To use an image or shape as a background, follow the steps below:

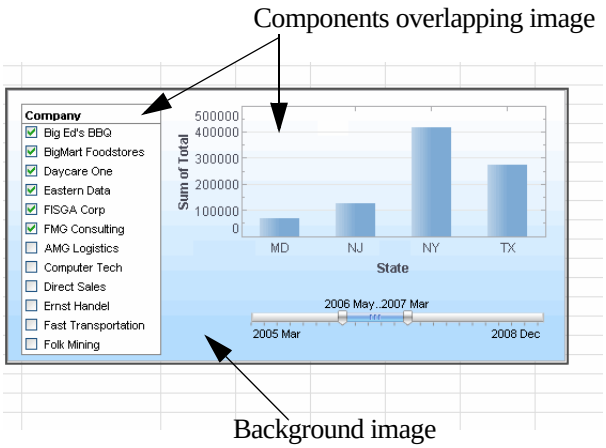
1. Add an image component or shape component to the Viewsheet.
2. For an image component, follow the steps below:
    a. Right-click the image component, and select ‘Properties’ from the context menu. This opens the ‘Properties’ dialog box.
    b. Select the **General** tab. From the ‘Static Image’ panel, specify an image to display as the background. (See *Image* for more details.)
    c. Select the **Advanced** tab, and enable the ‘Scale Image’ option.
    d. Click ‘OK’ to exit the dialog box.
3. Right-click the image or shape component, and select ‘Send to Back’ from the context menu.
4. Drag the image or shape into the desired position (usually behind another component or components).



5. Resize the image or shape as desired.



You can continue placing other components on top of this background image.



See Also

Moving a Component, for information on resizing an element.
Scaling an Image, for methods of scaling an image without distortion.
Shape Components, for detailed information about setting shape properties.
Image, for detailed information about setting image properties.

1.3.6 Previewing a Viewsheet

To preview a Viewsheet, do one of the following:



- Press the 'Preview' button in the main toolbar.
- Right-click in an empty region of the Viewsheet grid, and select 'Preview' from the context menu.

This opens a new **Preview** tab that presents the Viewsheet with full interactivity, just as end-users will experience it in the Portal.

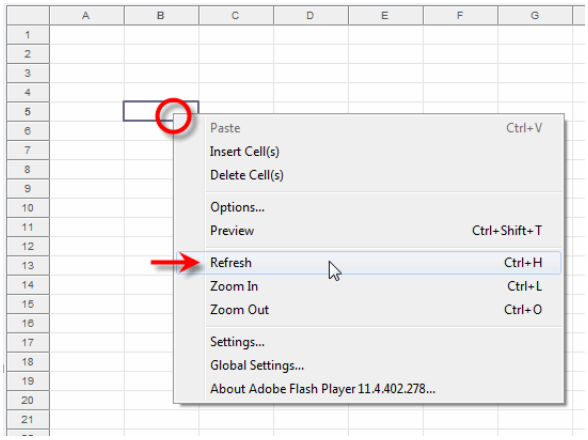
The design view must remain open for automatic refresh.

If you preview a Viewsheet, and then (without closing the **Preview** tab) return to the design view to make additional changes, you can refresh the existing Preview with your changes by simply clicking the **Preview** tab

again. You can also click the ‘Refresh’ button in the toolbar. (The ‘Refresh’ button is not available for materialized Viewsheets.)

1.3.7 Refreshing a Viewsheet

To refresh a Viewsheet while in design view, right-click in an empty region of the Viewsheet grid and select the ‘Refresh’ option from the context menu.



This refreshes the Viewsheet to reflect changes to the underlying data source and embedded components. To refresh a Viewsheet while in preview, see *Previewing a Viewsheet*.

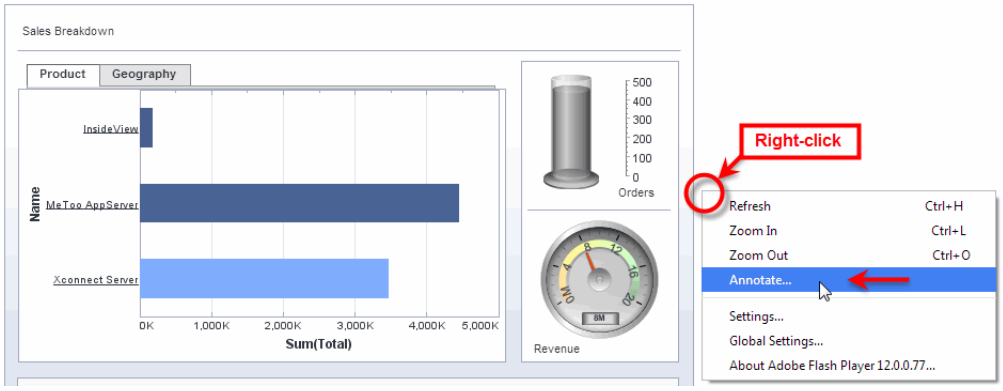
1.3.8 Adding Annotations to a Viewsheet

You can add annotations to the following parts of a Viewsheet:

- Viewsheet body
- Output and data view components
- Chart data point
- Table data cell

To add an annotation to a Viewsheet, follow these steps:

1. Preview the Viewsheet. See *Previewing a Viewsheet* for details.
2. If necessary, select the bookmark in which you want to save the annotation. (Annotations that you add will be saved with the active bookmark.) See *Restoring a Bookmark* for information on selecting a bookmark.
3. Right click on the Viewsheet body, a Viewsheet component, or a data point that you want to annotate, and select ‘Annotate’ from the context menu.



4. Add the desired annotation, and press ‘OK’.

See *Annotating a Dashboard* for complete information on adding, editing, and formatting annotations.

1.4 Deploying a Viewsheet

You can deploy a Viewsheet to the User Portal in a number of ways, and Viewsheets can be synchronized so that the data presented by one will reflect the selections made in others.

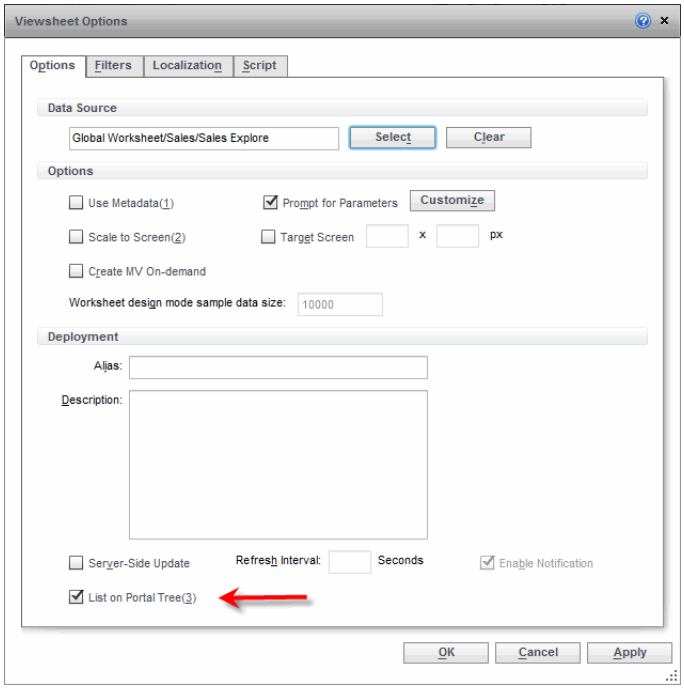
You can directly deploy a Viewsheet to the User Portal, where it will be accessible in the Repository tree under the **Report** tab. You can also deploy a Viewsheet under the **Dashboard** tab.

1.4.1 Deploying a Viewsheet to the Repository

To list a Viewsheet in the Portal Repository tree, follow these steps:

1. Click the ‘Options’ button in the toolbar. This opens the ‘Viewsheet Options’ dialog box.

2. Select ‘List on Portal Tree’.



3. Save the Viewsheet.

The accessibility of the Viewsheet to different users depends on the scope under which you save the Viewsheet.

See Also
Saving a Viewsheet, for information on setting Viewsheet scope and permission.

1.4.2 Deploying a Viewsheet as a Dashboard

To deploy a Viewsheet under the **Dashboard** tab of the Portal, see the appropriate section below:

- *Dashboards*, in *Advanced Server Administration*, to deploy a dashboard as an administrator.
- *Deploying a Dashboard* to deploy a dashboard as a user.

See Also
Interacting and Collaborating, for information on using dashboards.
Building a Composite Dashboard, for information on Viewsheet nesting.
Dashboards, in *Advanced Server Administration*, to configure and manage dashboards.
Setting Permissions, in *Advanced Server Administration*, for security configuration.

1.4.3 Building a Composite Dashboard

Some dashboards require a large number of input and output controls, and would be complex to design using just a single Viewsheet. In such cases, it is often easier to use a *modular* design, and build the final dashboard from a set of simple Viewsheets. Each of the simple Viewsheets might contain just a few controls, and provide specific functionality to the composite dashboard.

Modular designs offer benefits in terms of reusability, because you can deploy your simple Viewsheet modules in multiple dashboard applications. For example, you can create a Viewsheet that contains a simple cluster of Gauges, and then just plug that Viewsheet into any dashboard to provide the dashboard with that Gauge cluster.

Modular design also simplifies maintenance. When you use a Viewsheet in multiple composite dashboards, you can update all of the composite dashboards simultaneously by making changes centrally to the shared module.

To create a composite dashboard, simply nest a set of simple Viewsheets within a parent Viewsheet. The following sections explain how to do this, and how to synchronize the nested Viewsheets to work as a unified whole.

Nesting a Viewsheet

To embed or nest one Viewsheet within another Viewsheet, follow these steps:

1. Create the Viewsheet that you wish to embed. (This is the Viewsheet containing the common components that you wish to embed in other Viewsheets.)

2. Save this common Viewsheet to the Repository (under ‘Global Viewsheet’ or ‘User Viewsheet’, as desired.)

3. Open or create the host Viewsheet. (This is the Viewsheet within which the common Viewsheet will be embedded.)

4. Open the **Asset** panel on the left side of the Visual Composer to display the asset tree.

5. Drag and drop the common Viewsheet (saved in Step 2) from the Repository into the host Viewsheet.

The common Viewsheet is now embedded within the host Viewsheet, and all of the components marked ‘Visible in External Viewsheets’ within the common Viewsheet are displayed. To synchronize selection and input components within two (or more) Viewsheets, see *Sharing Filters*.

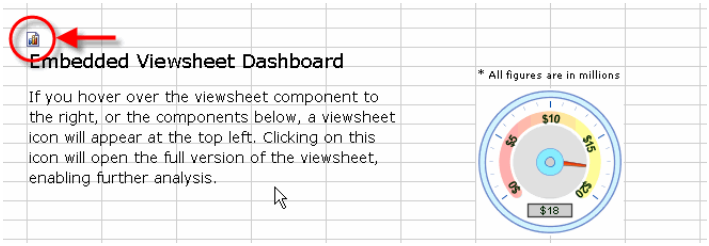
See Also

General Tab, for information on marking components as 'Visible in External Viewsheets'.

Nesting Viewsheets, for information on nested Viewsheet properties.

Opening a Nested Viewsheet

To open an embedded Viewsheet for editing, move the mouse over the embedded area in the host Viewsheet, and then click the Viewsheet icon in the upper left corner of the area.



This feature is also available to end-users who view the host Viewsheet in the Portal if there is at least one hidden element in the embedded Viewsheet. (See *Setting a Component to be Visible within External Viewsheets* for more information about hiding components.)

Configuring Interactivity among Nested Viewsheets

You can create interactivity between nested Viewsheets by synchronizing their selection elements. See *Sharing Filters* for more information.

1.4.4 Sharing Filters

You can *synchronize* selection and input components within a Viewsheet or across multiple Viewsheets so that a selection in one component is automatically propagated to other compatible components. This is called *sharing filters*. The following selection and input components can be synchronized as shared filters:

– *Range Slider*
– *Selection List*
– *Selection Tree*
– *Calendar*
– *Radio Button, Check Box, Combo Box*
– *Slider and Spinner*
– *TextInput*

There are four general cases where sharing filters is useful:

Note: Selection elements based on the same Data Block in the same Viewsheet are always automatically synchronized.

1. A Viewsheet displays data from multiple Data Blocks, and two or more of these Data Blocks share a common field that you wish to simultaneously filter.

   For example, consider a Viewsheet which has two charts, one presenting data from Data Block A and the other presenting data from Data Block B. Assume that each of these Data Blocks contains an 'Employee' field. In order to allow the user to filter both charts based on their respective 'Employee' fields, you need to create one Selection List for each Data Block. However, these two 'Employee' Selection Lists are not synchronized by default, which means that the user is free to select different values in each list. In this case, synchronize the two elements as described in *Sharing Filters Among Viewsheets*, and then hide one of the two selection components (set 'Visible' property to 'Hide').

2. A "parent" Viewsheet contains a nested "child" Viewsheet, and the child Viewsheet contains elements that should be filtered by selections the user makes within the parent Viewsheet. (See *Nesting Viewsheets* for information on how to embed one Viewsheet within another Viewsheet.)

   To synchronize elements across Viewsheets you must specify a shared filter, as described below, even if the elements utilize the same Worksheet Data Block.

   For example, consider a Viewsheet that contains a chart displaying 'Total' vs. 'Employee' and also contains a Selection List to filter the 'Employee' field. Nested within this parent Viewsheet is a child Viewsheet that contains a chart displaying 'Quantity' vs. 'Employee' and also contains a Selection List to filter the 'Employee' field. The two 'Employee' Selection Lists (one in the parent and one in the child) are not synchronized by default, which means that the user is free to select different values in each list. In this case, synchronize the two elements as described in *Sharing Filters Among Viewsheets*, and then disable the 'Visible in External Viewsheets' property of the Selection List in the child Viewsheet. (This prevents the child's Selection List from appearing in the parent Viewsheet, avoiding duplication of components).

3. A "parent" Viewsheet contains hyperlinks to a "child" Viewsheet, and the child Viewsheet contains elements that should be filtered by selections the user makes within the parent Viewsheet. (See *Hyperlinks* for information on configuring hyperlinks.)

   For example, consider a parent Viewsheet that contains a chart displaying 'Total' vs. 'Employee' and also contains a Selection List to filter the 'Employee' field. Assume that this parent Viewsheet contains hyperlinks to a child Viewsheet which contains a chart displaying 'Quantity' vs. 'Employee' and also contains a Selection List to filter the 'Employee' field. The two 'Employee' Selection

Lists (one in the parent and one in the child) are not synchronized by default, which means that the user is free to select different values in each list. In this case, synchronize the two elements as described in *Sharing Filters Among Viewsheets*.

Sharing Filters Among Viewsheets

To synchronize *Selection Components* and *Input Components*, assign them the same 'Shared Filter' ID. Only elements that have the same data type and same data binding should be synchronized. Follow the steps below:

1. Open the Viewsheet in the Visual Composer



2. Click on the 'Options' button in the toolbar. This opens the 'Viewsheet Options' dialog.

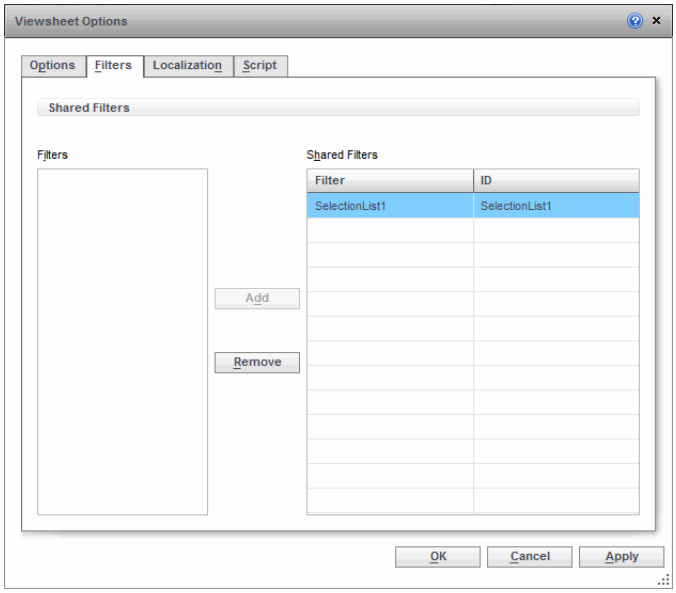
3. Select the **Filters** tab.

4. In the 'Filters' list, click on a selection or input component that you would like to synchronize, and drag it into the 'Shared Filters' list. (You can also double-click the item to add it to the list.)

```
Components that have the same ID (within the Viewsheet or across
Viewsheets) will be synchronized.
```

5. Click on the component label in 'ID' column of the 'Shared Filters' list, and enter a suitable name.

6. Repeat the above steps to add additional components to the 'Shared Filters' list.



7. Press the ‘OK’ button to exit the dialog box.

8. Repeat the same process for other Viewsheets (nested Viewsheet or hyperlinked Viewsheets).

Use identical IDs for components you wish to synchronize with one another. For example, to synchronize an ‘Employee’ Selection List in a “parent” Viewsheet with an ‘Employee’ Selection List in a nested “child” Viewsheet, follow the steps below:

1. Open the ‘Options’ dialog box for the *parent* Viewsheet. Assign the ‘Employee’ Selection List in the *parent* Viewsheet a shared filter ID such as “SharedEmployeeFilter”, and press ‘OK’.

2. Open the ‘Options’ dialog box for the *child* Viewsheet. Assign the ‘Employee’ Selection List in the *child* Viewsheet the same shared filter ID (“SharedEmployeeFilter”), and press ‘OK’.

This will synchronize the ‘Employee’ Selection Lists in the two Viewsheets.

Sharing Filters Between Viewsheets and Reports

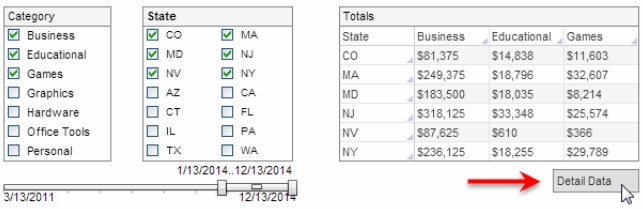
To share filters between a dashboard and report, follow the steps below:

1. Assign filter IDs to the desired dashboard components as described in *Sharing Filters Among Viewsheets*.

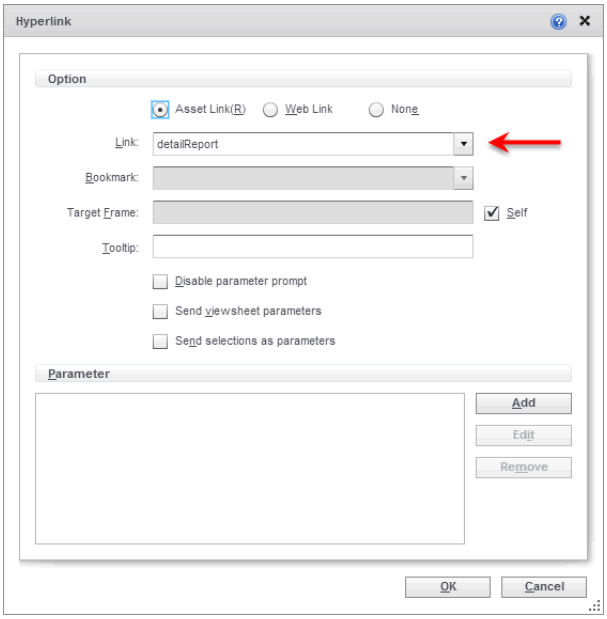
2. Assign filter IDs to the desired report data fields as described in *Sharing Filters with a Dashboard* in *Advanced Report Design*.

When you create a set of shared filters between a dashboard and report, the report will be filtered to match the dashboard selections whenever a user opens the report while the dashboard is open.

In most cases where shared filters are used, the user opens the report by clicking a hyperlink in the dashboard. For example, the dashboard below provides a hyperlink on the ‘Detail Data’ text component.



The hyperlink targets the desired detail report.



See *Hyperlinks* for information on how to add hyperlinks to text, images, and other components.

1.4.5 Enabling End-User Chart and Crosstab Editing

To provide end users with access to the Chart Editor and Crosstab Editor, follow the steps below:

1. Right-click the desired chart or crosstab and select 'Properties' from the context menu. This opens the 'Chart Properties' or 'Crosstab Properties' dialog box.

2. Select the **Advanced** tab in the dialog box.

3. Select the 'Enable Ad Hoc Editing' option.

If this option is not selected, users will not have access to the Chart Editor or Crosstab Editor. However, other features (brushing, show details, export, zooming, etc.) will remain available.

See Also
Creating a Chart, for information on designing with the Chart Editor.
Creating a Crosstab, for information on designing with the Crosstab Editor.
Interacting and Collaborating, for information on working with dashboards.

1.4.6 Setting Default Viewsheet State

When you save a Viewsheet using the Visual Composer, the current state of the Viewsheet (selection choices, menu choices, etc.) is saved as the

Viewsheet's *default bookmark*. The default bookmark defines the default state of the Viewsheet when it is first viewed by the user. (See *Bookmarking a Dashboard* for more information about bookmarks). Therefore, to set the default Viewsheet state, simply arrange the Viewsheet controls as desired in Visual Composer, and then save the Viewsheet.

See Also
Saving a Viewsheet, for information on saving a Viewsheet.

1.4.7 Localization/Internationalization

If the dashboards you create are viewed from multiple locales, it may be desirable to configure Viewsheet components to display text in a language specific to the user locale. The following Viewsheet components can be localized in this way:

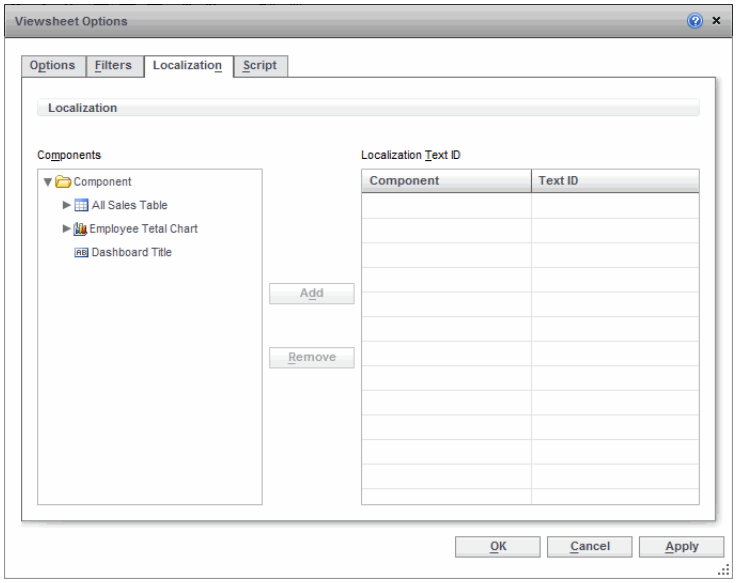
- Table title and column header text
- Freehand Table title and text cell text
- Crosstab title, column header text, and grand total text
- Chart axis titles and legend titles
- Selection component titles
- Text component text
- Input component titles
- Submit button label

You can configure a Viewsheet component for localization by associating one or more Text ID properties with the component. The specified Text IDs for each component are then used to create a mapping between the text string(s) in the component and the replacement text string(s) to be shown to the user.

To specify a Text ID for a component, follow the steps below:

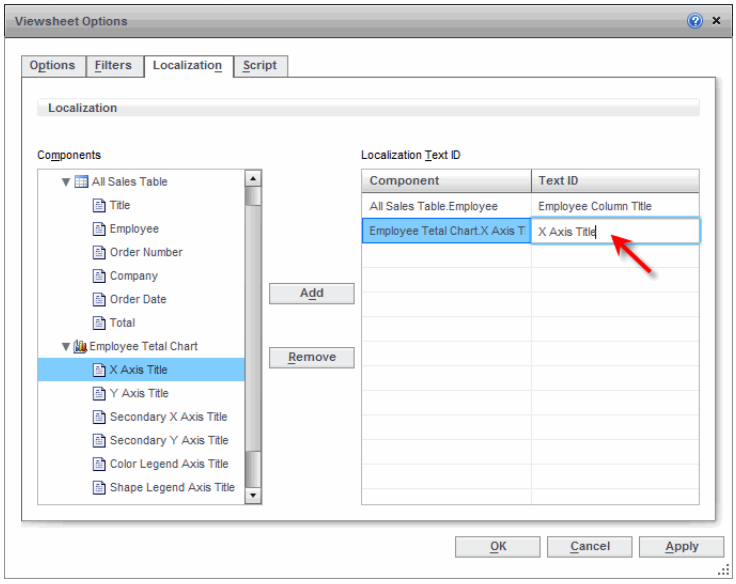
1. Press the 'Options' button in the Visual Composer toolbar to open the 'Viewsheet Options' dialog box.

2. Select the **Localization** tab.



3. In the left ‘Component’ panel, select the property in the element tree that you wish to localize, and press ‘Add’. This adds the property to the right ‘Localization Text ID’ panel.

4. In the ‘Localization Text ID’ panel, enter a label for the element in the ‘Text ID’ field.



The Text ID that you enter here is the label used to represent the text of the corresponding component in the localization mapping file discussed below.

5. Press ‘OK’ to exit the ‘Viewsheet Options’ dialog box.

The mapping between the specified Text IDs and the replacement text for each locale is specified in a resource bundle specific to that locale. The resource bundle is a "*.properties*" file, and must be available on the classpath, preferably in the *sree\WEB-INF\classes* directory. The file should be named using the ISO-standard two-letter codes for language and country; for example, a property file supporting Canadian French speakers would be called '*SreeBundle_fr_CA.properties*'. See the *Localization/ Internationalization* section of the *Advanced Server Administration* for details on configuring resource bundle contents.

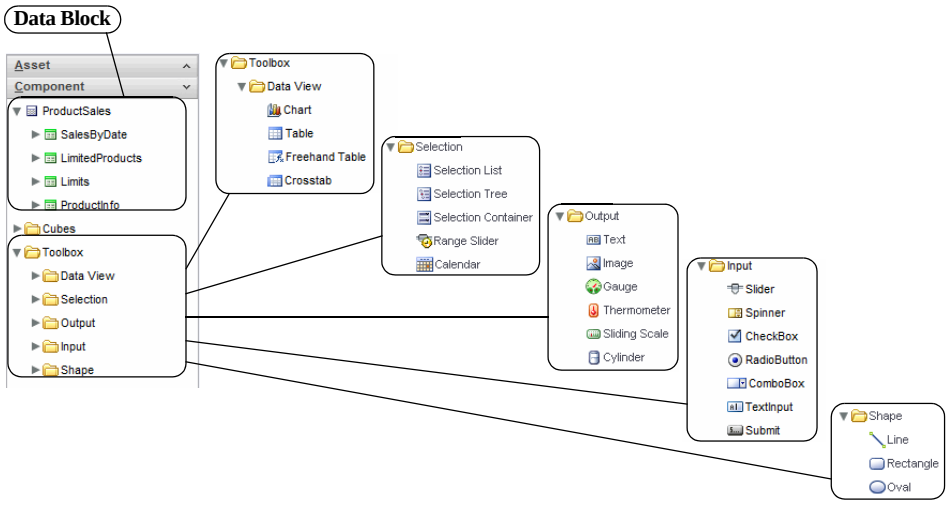
When the end-user selects a locale from the User Portal login menu, the mapping in the corresponding resource bundle is used to display the replacement text for all mapped elements. If a resource bundle corresponding to the user's locale selection is not found on the classpath, no localization will be performed.

In addition to localizing Viewsheet elements, you can also configure localization for User Portal interface components (tabs, tree elements, etc.). See the *Localization/Internationalization* section of the *Advanced Server Administration* for instructions.

2 Using Viewsheet Components

Viewsheet components are listed under the 'Toolbox' node of the Component tree, on the left side of the Visual Composer. Viewsheet components share a number of common features, including certain properties, formatting options, and hyperlinks. This section explains these common features.

For options that are unique to a specific component, see the related section in this guide.



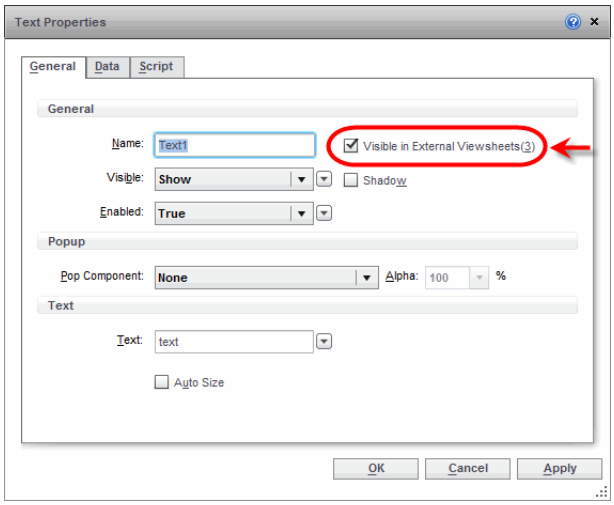
See Also

Data View Components
Selection Components
Output Components
Input Components
Shape Components
Organization Components

2.1 Common Operations

The component context menu provides access to certain common operations discussed in this section.

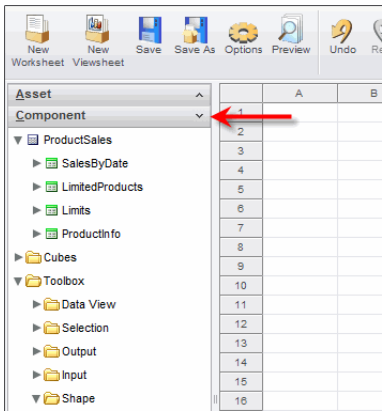
Figure 3. The component right-click context menu.



2.1.1 Adding a Component

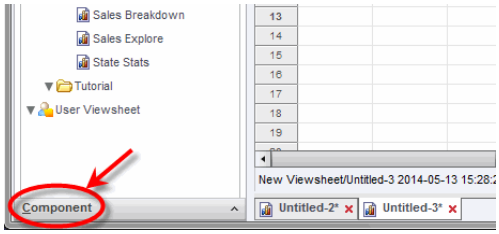
To add a component to a Viewsheet, follow these steps:

1. Open the **Component** panel on the left-side of the Visual Composer.



When no Viewsheet is open for editing, the components in the **Component** panel appear dimmed (disabled). In this case, verify that you are editing Viewsheet, not a Worksheet.

If the **Component** panel is not visible (i.e., the **Asset** panel is showing), click the **Component** title bar at the bottom of the screen.



2. Expand a node on the ‘Toolbox’ tree (Data View, Selection, Output, Input).

3. Click and drag the desired component onto the Viewsheet grid. If there is sufficient grid space, this creates the new component on the grid.

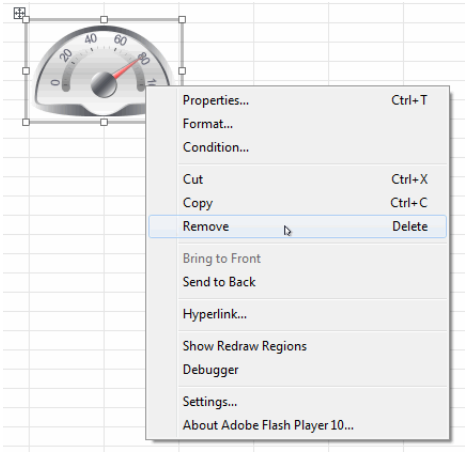
   If there is not sufficient space on the grid to add the new component (due to other components nearby), the ‘Layout Options’ dialog box will open. This allows you to specify the method for repositioning nearby components to create space for the new component, or to overlap the exiting component on top of the exiting components. See *Overlapping Components* for details about the available options.

In many cases, after you add a component, you will need to *bind* the component to a data field in the Data Block. For information on how to do this, see the individual Component section.

See Also
Data View Components, for Table, Chart, Crosstab.
Selection Components, for Selection List/Tree, Range Slider, Calendar.
Output Components, for Text, Image, Gauge, Thermometer, Scale, Cylinder.
Input Components, for Slider, Spinner, CheckBox, RadioButton, ComboBox.

2.1.2 Deleting a Component

To delete an element from the Viewsheet grid, right-click the component and select ‘Remove’ from the context menu (or press ‘Delete’ on the keyboard).



You can also cut-and-paste the component so that it deleted from one location and inserted at another.

See Also

Moving Components using Cut and Paste, for information on cut, copy, and paste.

Properties Dialog Box, for information on hiding elements.

Setting a Component to be Visible within External Viewsheets, for information on hiding nested Viewsheets.

2.1.3 Positioning a Component

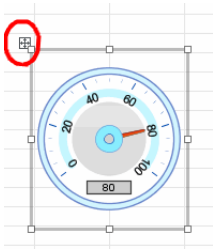
To position a components in a Viewsheet, you can either drag the component to the desired location, or cut and paste the component. To change the size of a component, drag the resizing handles on the component's borders. The following sections discuss these methods in more detail.

Moving a Component

To move a component by dragging with the mouse, follow these steps:

To move a table, drag the title bar. To move an output component, drag its body.

1. Click the component to select it. This enables the "move" handle at the top left corner.



2. Click and drag on the "move" handle to move the component to the desired location. (Hold down the 'Alt' key while dragging to snap the component to the Viewsheet grid.)

You can also use the keyboard arrow keys to move the selected component. Hold down the 'Ctrl' key together with the arrow keys to move the component in smaller increments.

```
When exporting a Viewsheet to Excel, grid-aligned Text elements are
exported as editable values. Non-aligned Text elements are exported
as images.
```

Most components can be positioned and resized freely, independent of the Viewsheet grid. However, the following components always snap the Viewsheet grid: Checkbox, Radio Button, all Table types, Selection Lists/ Trees, and Groups.

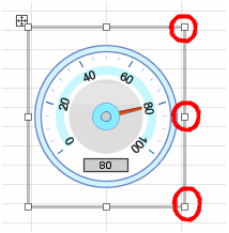
See Also

Grouping Components, for information on creating component groups.

Resizing a Component

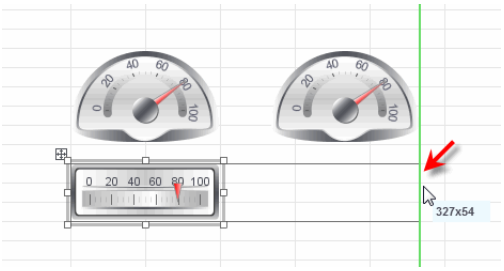
To resize a component, follow these steps:

1. Click the component to select it. This enables the component "resize" handles.



2. Click and drag on a resize handle or on the component border to resize the component. (Hold down the 'Alt' key while dragging to snap the component border to the grid.)

   To align the border with the border of another component, drag until you see the green alignment guide, and then release.



Most components can be positioned and resized independent of the Viewsheet grid. However, the following components always snap the Viewsheet grid: Checkbox, Radio Button, all Table types, and Groups.

Components such as Charts, Tables, and Selection Lists display a scroll bar if the component is not large enough to display all information in a single view.

See Also

Zooming the Viewsheet Grid, to resize all Viewsheet components uniformly.

Grouping Components, for information on creating component groups.

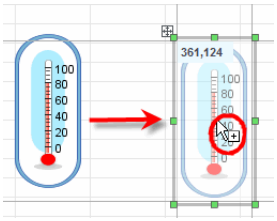
Moving Components using Cut and Paste

Tip: To copy the contents of a Text element, double-click the element.

To cut or copy a component, right-click the component and select ‘Cut’ or ‘Copy’ from the context menu. (Alternatively, click Ctrl-X or Ctrl-C on the keyboard).

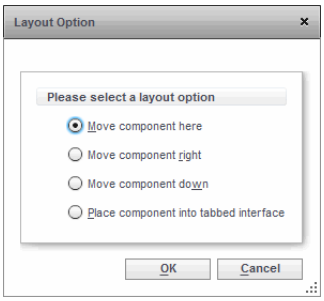
To insert the component elsewhere in the Viewsheet, right-click the desired region and select ‘Paste’ from the context menu (Ctrl-V on the keyboard). If other components already occupy this region, the inserted component will overlap those components. See *Overlapping Components* for more information.

You can copy and move a component with a single action. Simply Ctrl-click the element, and drag it to a new location. The cursor displays a ‘+’ sign while dragging to indicate that a copy is being created.



Overlapping Components

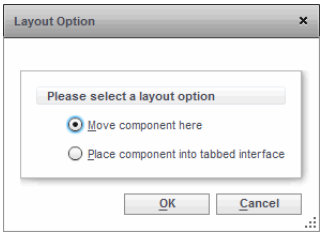
When you drag a new component from the Component tree onto an existing Viewsheet component, you will see the ‘Layout Option’ dialog box:



The following options are available:

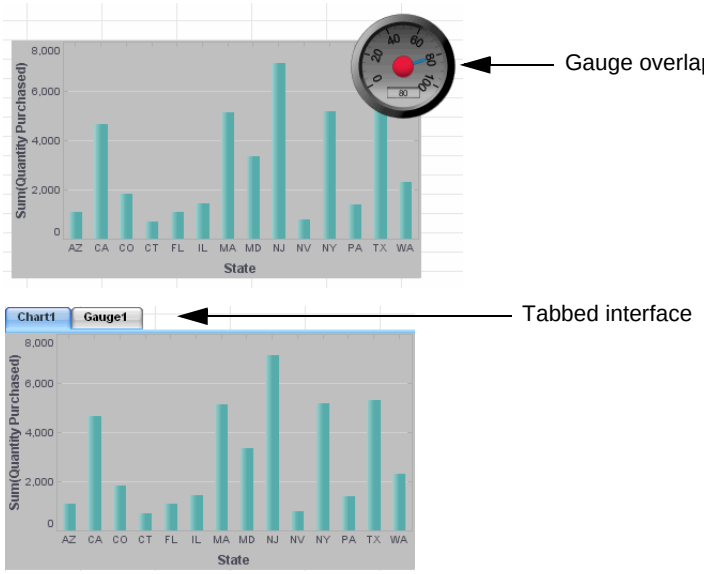
- ‘Move component right’ places the new component in the first available column to the right of your original placement (same row).
- ‘Move component down’ places the new component in the first available row below your original placement (same column).
- ‘Place component into tabbed interface’ creates a tabbed interface with each component in its own tab.

When you move or paste an *existing* component onto another existing component, you will see a slightly different version:



The following options are available:

- ‘Move component here’ permits the components to naturally overlap each other.
- ‘Place component into tabbed interface’ creates a tabbed interface with each component in its own tab.



To move an overlapping component behind other components, right-click the component and select ‘Send to Back’ from the context menu. To move an overlapping component in front of other components, right-click the component and select ‘Bring to Front’ from the context menu.

See Also

Creating a Tabbed Interface, for information on using placing components in tabs.

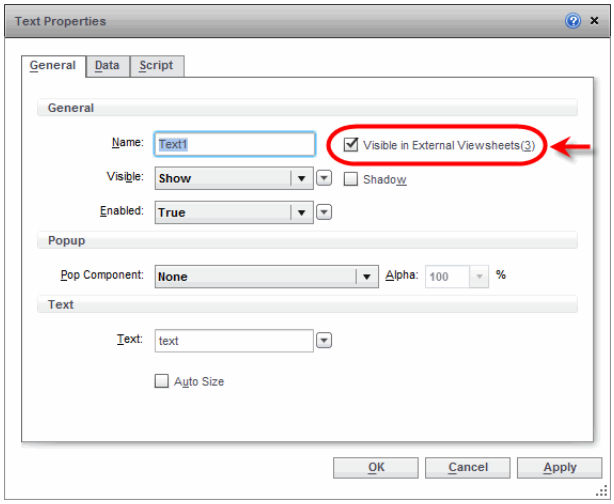
Setting Colors, for information setting transparency.

2.1.4 Setting a Component to be Visible within External Viewsheets

To set a component to be visible within an external Viewsheet (i.e., within a Viewsheet that hosts the component's Viewsheet as an embedded Viewsheet), follow the steps below:

1. Right-click the element and select 'Properties' from the context menu. This opens the 'Properties' dialog box.

2. Enable the 'Visible in External Viewsheets' option in the dialog box.



3. Click 'OK'.

All elements set as 'Visible in External Viewsheets' will be visible (unless explicitly hidden) when the Viewsheet is embedded (nested) within another Viewsheet.

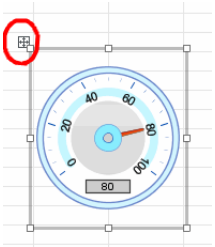
See Also

Building a Composite Dashboard, for information on embedding Viewsheets.

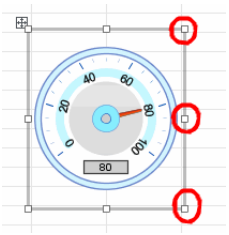
Properties Dialog Box, for information on element visibility.

2.1.5 Selecting Components

To select a single component on the Viewsheet, click on the component with the mouse. This activates the component's "move" handle and "resize" handles, which then allow you to position and size the component.



Move Handle



Resize Handles

To simultaneously select multiple components on the Viewsheet, use one of the following techniques.

In some cases you may need to Ctrl-click on the component title bar or "move" handle to deselect the component.

- **Ctrl-click** on each component in turn to add it to the selection. (Ctrl-click an already-selected component to remove it from the selection.)
- **Shift-click** on each component in turn to add it to the selection. (Ctrl-click an already-selected component to remove it from the selection.)
- Click in an empty region of the Viewsheet, and **drag the mouse** to draw a selection region covering the desired components. When you release the mouse, all components that are partially or wholly covered by the selection region become selected.

You can also use these techniques in combination with each other.

For example, by dragging the mouse to create the selection region below, you can easily select all three components. You only need to include part of each component within the selection region.



Selecting multiple components allows you to move the components together, group the components, and set common properties (e.g., font).

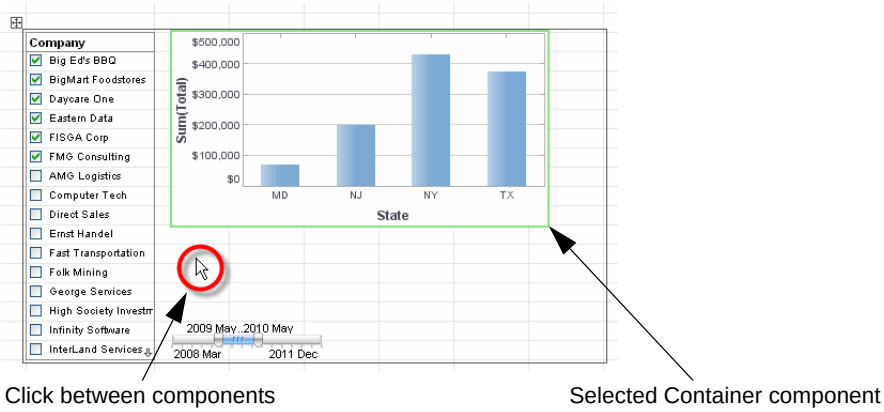
See Also

Grouping Components, for more information on keeping components together.

Positioning a Component, for more information on moving and resizing.

2.1.6 Selecting Groups and Grouped Components

To select the Container component itself, click on any component within the group, or click in any empty space between group components. This activates the Container's "move" handle and "resize" handles, which then allow you to position and size the Container component.



To select an individual component within a group, first select the group as above, and then click the desired component.

See Also

Organization Components, for complete information on group behavior.

2.1.7 Grouping Components

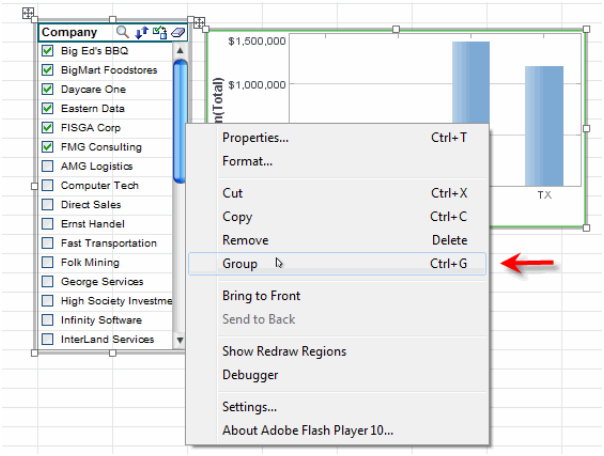
When you need to position a set of components together, it is helpful to combine the components into a *group*. Components within a group always retain their relative positions when the group is moved, and cannot be positioned independently.

```
Note: A group always snaps to the Viewsheet grid. Selection
Containers cannot be grouped.
```

To create a group, follow these steps:

1. Select the components you wish to group. (See *Selecting Components*.) You can also select existing groups.

2. Right-click any selected component, and choose 'Group' from the context menu (or press Ctrl-G on the keyboard).



This creates a new Container component to contain the selected components. The components within a Container remain fully functional, and you can adjust the component settings and properties in the usual manner.

A Container component contains only individual components, not other Container components. Therefore, if you group together two existing groups, the existing Containers are deleted, and a single new Container is created to contain all of the individual components from the two preexisting groups. See *Organization Components* for complete information about groups.

To undo a grouping, right-click the group and select 'Ungroup' from the context menu (or press Ctrl-Shift-G on the keyboard).

See Also

Organization Components, for complete information on group behavior.

Building a Composite Dashboard, to nest or embed one Viewsheet in another.

Creating a Tabbed Interface, to provide tab access to overlapping components.

2.2 Properties Dialog Box

To set the properties of a Viewsheet component, right-click the component in the Viewsheet grid, and select 'Properties' from the context menu (or press Ctrl-R on the keyboard). This opens the 'Properties' dialog box for the component.

To set the properties of a Container component, right-click in an empty region between the components in the group, and select 'Properties' from the context menu. This opens the 'Properties' dialog box for the Container component.

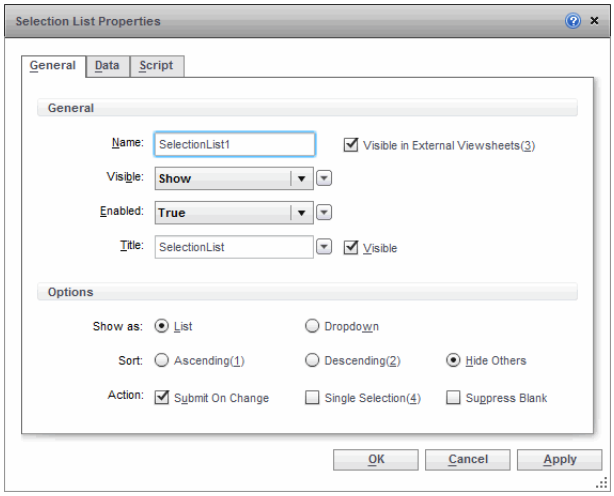
Many of the properties in the 'Properties' dialog box are common across components. The following sections discuss these common properties. Properties that are unique to a particular element are discussed in the section for that element.

See Also
Table Properties
Crosstab Properties
Chart Properties
Map Chart Properties
Selection List Properties
Selection Tree Properties
Range Slider Properties
Calendar Properties
Slider and Spinner Properties
Radio Button/Check Box/Combo Box Properties
Container Properties
Selection Container Properties
Shape Properties

2.2.1 General Tab

The **General** tab of the 'Properties' dialog box is common to many components. It has five standard properties, shown in the table below.

Name	This name is used to refer to the component in binding and scripting. For purposes of scripting, the name should contain no spaces.
Visible in External Viewsheets	If checked, the component will be visible when this Viewsheet is embedded in another Viewsheet (see *Building a Composite Dashboard*).
Visible	**Show**: Element is visible. **Hide**: Element is invisible. **Hide on Print and Export**: Element is visible on screen but hidden when exported. (
Enabled	**true**: Element is interactive. **false**: Element is not interactive. Note: For Selection components, when 'Enabled' is 'false', existing selections in the component do not affect Viewsheet data.
Title	The text that appears in the title bar (for components which have one). The 'Visible' option determines whether the title bar itself is displayed. Besides the 'Title' field, you can also edit title bar text by double-clicking the title bar itself.



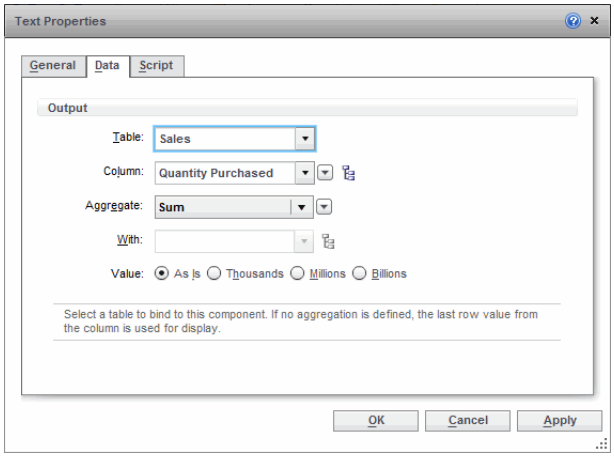
2.2.2 Data Tab

Note: You can also associate data with a component by dragging a field from the data source onto the component. See *Creating a Dashboard* for an example.

The **Data** tab in the 'Properties' dialog box is common to many components. This tab allows you bind the component to the data source that supplies the data displayed by the component. The **Data** tab has the standard properties shown below.

Table	The query, model, table, or Worksheet Data Block to which the component is bound.
Target	For input components (Radio Button, Spinner, etc.), the Embedded Table or Variable in the Worksheet to which the input value is passed. (See *Using Input Components* for further details.)
Column	The field in the selected table that supplies data to the component.
Aggregate	The method of data aggregation/summarization. (See *Aggregation Options* for a description of the methods.)
With	A second data field for aggregation methods requiring two operands.
Value	The scaling factor used to display data on output components: **As Is**: No scaling. **Thousands**: Displayed values are divided by 10^3. **Millions**: Displayed values are divided by 10^6. **Billions**: Displayed values are divided by 10^9.

`Note: The` **`Data`** `tab has a different layout for Selection components. See` *`Selection Components`* `for more details.`



2.2.3 Advanced Tab

The **Advanced** tab of the 'Properties' dialog box is common to many components, but options presented vary for different components.

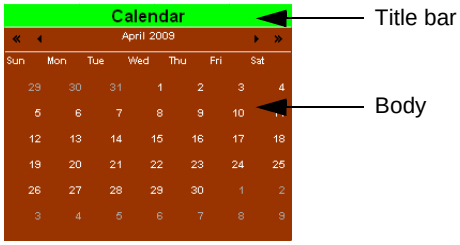
2.2.4 Script Tab

The **Script** tab of the 'Properties' dialog box is common to all components, and contains a Script Editor that allows you to insert component-level scripts. These scripts can modify component properties based on programmatic logic. See *Advanced Dashboard Scripting* for complete information about scripting. capabilities.

2.3 Format Dialog Box

The 'Format' dialog box provides options for changing the visual appearance of elements and their representation of data. To open the 'Format' dialog box, right-click on a component and select 'Format' from the context menu.

Certain components have multiple regions that can be selected and formatted independently. For example, a component title bar can always be formatted independently of the component body.



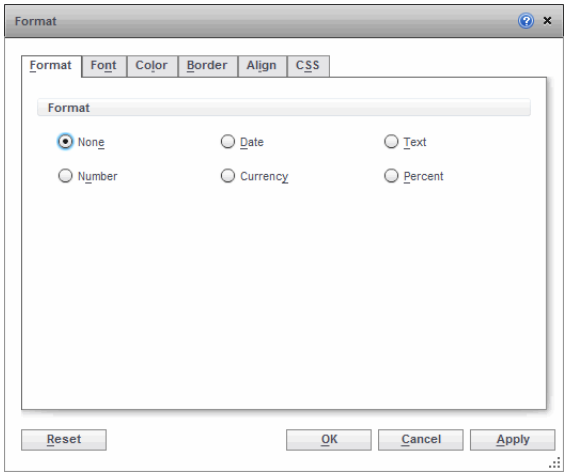
A more specific format overrides a more general format. For example, a title bar format overrides a general component format.

To format a particular region of a component, right-click on the desired region, and select ‘Format’ from the context menu. To format the general component (rather than a particular region), first make sure the component is entirely *deselected*. Then right-click to select the general component (not a region), and choose ‘Format’ from the context menu.

The table below lists the regions that can be independently formatted on multiple-region components:

COMPONENT	SELECTABLE REGIONS
Table	Title bar, column header, column body cells
Crosstab	Title bar, column headers, row headers, summary cells
Chart	Axis titles, axis labels, data label, legend title, legend content, target label
Selection List	Title bar, body cells
Selection Tree	Title bar, nth-level body cells (levels are independent)
Selection Container	Title bar
Calendar	Title bar, year labels, month labels, day labels

The ‘Format’ dialog box contains the following tabs: **Format**, **Alignment**, **Font**, **Border**, **Color**, and **CSS**. The following sections describe each of these tabs.

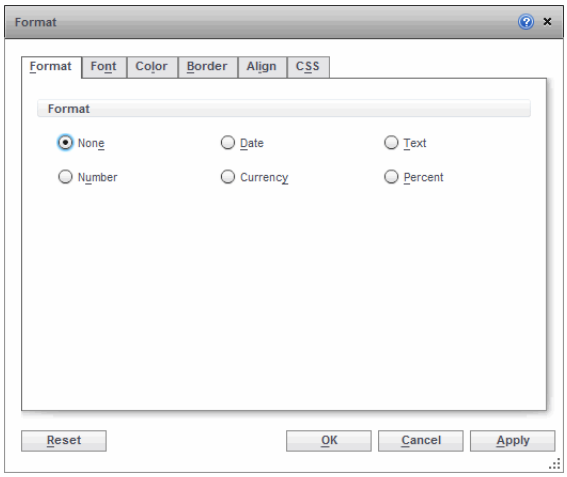


2.3.1 Format Tab

The **Format** tab allows you to specify a data format. Formatting is available for the following data types: Date, Number, Currency, Text, Percent.

A data format is a method for representing data as a string (for display purposes). For example, the date December 31, 2008 might be displayed as “12/31/08” or “Dec. 31, 2008” or simply “2008,” depending on the Date

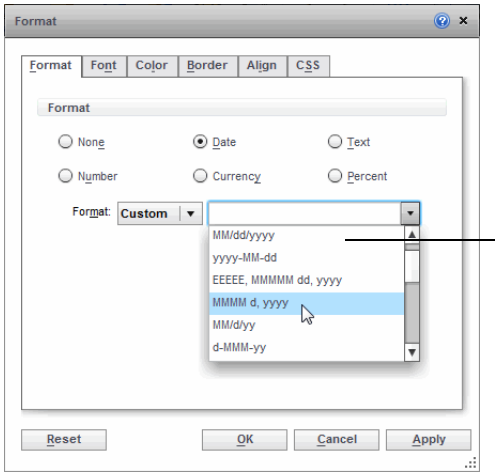
format specified. Likewise, the number 123456 might be displayed as “123,456” or “123456.00” or “00123456,” depending on the Number format specified.



The following sections explain these available formats.

Date Format

For ‘Date’ formatting, several predefined formats are available from the pop-up menu.



Predefined Formats

Date masks follow the *java.text.SimpleDate-Format* format.

To create other formats, use the following date masks:

```
M = Month
d = date
y = year
E = day of the week
```

Example: Date Format

For the date Nov 8, 2006:

```
M = 11; MM = 11; MMM = Nov; MMMM = November
```

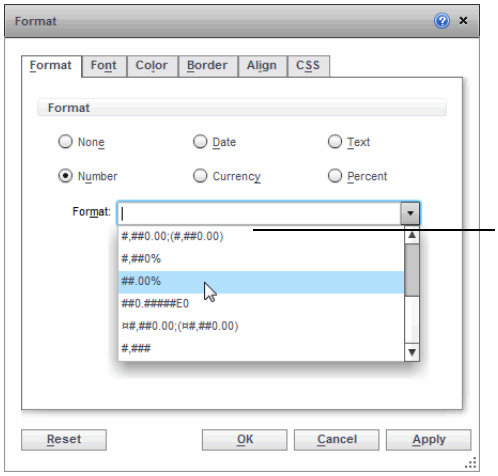
```
d = 8; dd = 08;
yy = 06; yyyy = 2006
EEE = Wed
EEEE = Wednesday
```

Therefore, the mask '`MMM-dd-yyyy`' yields the string "Nov-08-2006."

¢

Number Format

For 'Number' formatting, several predefined formats are available.

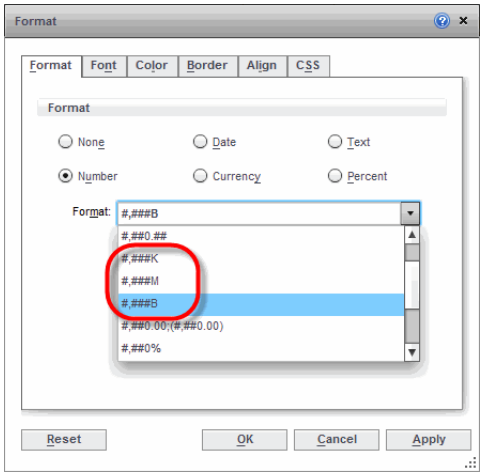


Predefined Formats

Decimal formatting only works if the value is truly a numeric type. If the numeric value you wish to format is actually text, convert the type using JavaScript so that formatting can be applied.

The number format pattern string uses the pattern defined by the java.text.DecimalFormat class. For example, to add commas to a number, use '#, ###, ###. ##'. Using a '#' will show a digit if needed, a '0' will display zero if the digit is not needed. Adding a '%' at the end of the format will multiply the value by 100 and display as a percentage.

You can add a suffix "K", "M", or "B" to the pattern string to automatically divide the numerical value by 10^3, 10^6, or 10^9, respectively.



To create custom multipliers, see *Defining Custom Number Format Multipliers* in *Advanced Server Administration*.

Example:
Number Format:

For the number 124521.63:

```
#,###.##  = 124,521.63
#,###.000 = 124,521.630
```

¢

Currency Format

The 'Currency' format represents the number as a currency rounded to two decimal places For example, the number 25.867 displays as $25.87. The currency symbol is automatically determined by the locale settings of the Operating System.

To apply a fixed currency symbol for all locales, use the 'Number' format option with a recognized currency symbol, e.g., '`$#,###.##`'.

Text Format

Use 'Text' format to add a prefix or suffix to string data. The special tag '`{0}`' acts as a place holder for the actual data. Use the escape sequence "`\n`" to insert a carriage return in the text.

Text Format Examples

1. Consider a text column containing the following values:

   ```
   Annie
   Eric
   Robert
   Sue
   ```

   The text format "`Employee name: {0}`" will display these values as follows:

   ```
   Employee Name: Annie
   Employee Name: Eric
   Employee Name: Robert
   Employee Name: Sue
   ```

2. Consider a text column containing the following values:

   ```
   NJ
   NY
   CA
   AZ
   ```

   The text format "`-{0}-`" will display these values as follows:

   ```
   -NJ-
   -NY-
   -CA-
   ```

-AZ-

Localizing a Text Format

To apply localization within a text format, use a string identifier in curly braces within the text format. The identifier in curly braces is used as a Text ID to locate the corresponding replacement text (usually a translation into another language) within the *SreeBundle* localization file. For example, the string "`{replacementText}`" within a text format will use "replacementText" as the Text ID.

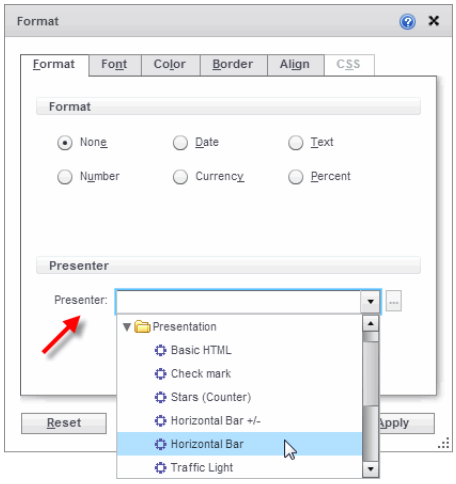
For information on how to map Text IDs to replacement text in the *SreeBundle* localization file, see *Localizing Reports, Viewsheets, Data Models/Queries* in *Advanced Server Administration*.

Percent Format

The 'Percent' format displays a number as a percent. For example, percent format displays the number 0.15 displays as "15%."

Presenter

A *presenter* is a graphical representation of a numeric value. You can assign a presenter to a numerical cell of a table or to a text element that displays numerical data.



The following table lists some commonly used presenters.

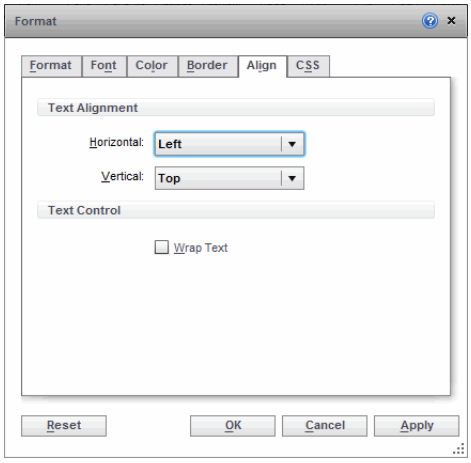
PRESENTER	DESCRIPTION
Basic HTML	Display strings with embedded tags, such as <b> for bold and <u> for underline
Checkmark	For a '`true`' Boolean value, displays a check mark. Otherwise, displays nothing.

Stars(Counter)	Draws icons (stars) horizontally. The number of icons is determined by value; for example, two icons are drawn for a value of two. **Options**: 'Icon' specifies the image used, and 'Icon Color' sets the background color.
Horizontal Bar +/-	Same as Horizontal Bar, but handles both positive and negative numbers. **Options:** 'Color' sets the positive bar fill color, 'Maximum' sets the value represented by a full-width bar, and 'Negative Color' sets the negative bar fill color.
Horizontal Bar	Draws a horizontal bar based on the numeric cell value, similar to a bar chart. **Options:** 'Color' sets the positive bar fill color and 'Maximum' sets the value represented by a full-width bar.
Traffic Light	Represents data ranges using three different colors. **Options**: 'Border' sets the border line style and color, 'Low' sets the color for the low range, 'Low Value' sets low-middle transition point, 'Mid' sets the color for the middle range, 'High' sets the color for the high range, 'High Value' sets the middle-high transition value, 'Low at Top/Right' indicates that the right or top light in the traffic light represents the low range (otherwise, the left or bottom light is the low range), 'Horizontal' indicates that the traffic light is displayed horizontally, 'Single Light' displays just one light instead of three, and 'Size' specifies the size of the traffic light.
Barcode	Display various bar codes. **Options**: 'Bar Height' sets the height of the bars in pixels, 'Bar Width' sets the width of the bars in pixels, and 'Resolution' specifies the code image resolution in pixels/inch.
2D Code	Two-dimensional QR Code for alphanumeric data. **Options**: 'Width' sets the width of the code in pixels.
Clockwise Rotation	Rotates contents by 90 degrees in clockwise direction.
Counterclockwise Rotation	Rotates contents by 90 degrees in counter-clockwise direction.
Multi-Header	
Button	Adds a 3D border. **Options**: 'Color' sets the button color, 'Bezel Width' sets the width of the button border.
Shadow	Adds a shadow. **Options**: 'Shading' sets the background color, 'Shadow Width' sets the thickness of the shadow, and 'Text Color' sets the font color.

2.3.2 Alignment Tab

You can also set alignments by using the Viewsheet toolbar.

The **Alignment** tab settings determine the horizontal and vertical alignment of text in Viewsheet components, including labels and titles on Charts. The 'Wrap Text' option wraps text to multiple lines instead of truncating the text.

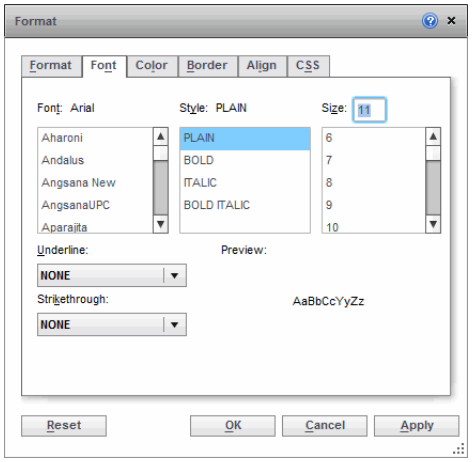


The text affected by the alignment setting varies for each component, as shown below:

COMPONENT	ALIGNMENT TARGET
Table	Alignment can be set independently for title text, column header text, and data row text. Select the title bar, column header, or data cell, and then select 'Format' from the context menu to set the corresponding alignment.
Embedded Table	Same as Table.
Crosstab	Same as Table.
Chart	Alignment can be set independently for axis titles, Y-axis labels, legend titles, and legend labels.
Selection List	Alignment can be set independently for title text and choice labels. Select either the title bar or a choice cell, and then select 'Format' from the context menu to set the corresponding alignment.
Selection Tree	Same as Selection List. Alignment can be set independently for each level of the hierarchy.
Selection Container	Alignment can be set on the component, and then overridden on each member selection.
Range Slider	Alignment setting affects all text displayed on the slider (range display and endpoints).
Calendar	Alignment setting affects the title text.
Text	Alignment setting affects the text
Slider	Alignment setting affects the labels.
Spinner	Alignment setting affects the spinner text.
Checkbox	Alignment can be set independently for title text and choice labels. Select either the title bar or a choice cell, and then select 'Format' from the context menu to set the corresponding alignment.
Radio Button	Same as Checkbox.
Combo Box	Alignment setting affects the choice text.

2.3.3 Font Tab

The **Font** tab allows you to specify the font face (Arial, Serif etc.), the style (bold, italic), and the size of the font used for component text.



The text affected by the font setting varies by component, as shown below:

COMPONENT	ALIGNMENT TARGET
Table	Font can be set independently for title text, column header text, and data row text. Select the title bar, column header, or data cell, and then select 'Format' from the context menu to set the corresponding font.
Embedded Table	Same as Table.
Chart	Font setting affects all the text, including axis labels, titles, legends, and text binding.
Crosstab	Same as Table.
Selection List	Font can be set independently for title text and choice text. Select either the title bar or a choice cell, then select 'Format' from the context menu to set the corresponding font.
Selection Tree	Same as Selection List. Font can be set independently for each level of the hierarchy.
Selection Container	Same as Selection List and Range Slider.
Range Slider	Font setting affects all text displayed on the slider (range display and endpoints).
Calendar	Font setting affects the title text.
Text	Font setting affects the text
Gauge	Font setting affects labels and value (if present).
Thermometer	Font setting affects the labels.
Sliding Scale	Font setting affects the labels.
Cylinder	Font setting affects the labels.
Slider	Font setting affects the labels.
Spinner	Font setting affects the spinner text.

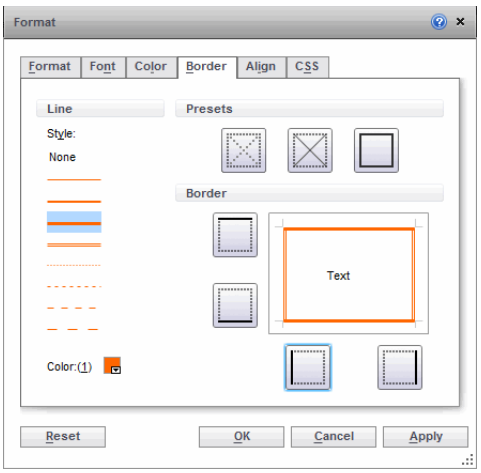
COMPONENT	ALIGNMENT TARGET
Checkbox	Font setting affects the choice labels. Select a choice cell, and then select ‘Format’ from the context menu to set the corresponding font.
Radio Button	Same as Checkbox.
Combo Box	Font setting affects the choice text.

See Also

Adding Highlights to a Table, for information on conditional font assignments.

2.3.4 Border Tab

The **Border** tab allows you to set border thickness and color for components.



Borders for Tables can also be set using Table Styles.

For most components, the border setting determines the appearance of the component’s boundary. For Tables, a border can be set for the table as a whole, or independently for the title bar, columns headers, and column cells.

Setting Borders

To set borders for a component, follow these steps:

1. Right-click the component, and select ‘Format’ form the context menu. This opens the ‘Format’ dialog box.

   For a Table, right-click on the region to which you want to add the border:

   a. Right-click a data cell to apply the border to all data cells in the column.

b. Right-click on the column header cell to apply the border to only the column header.

c. Right-click the title bar to apply the border to only the title bar.

d. Right-click the Table handle (with no cell selected) to apply the border to the Table's outer boundary. The selected border color is also applied to the cell boundaries.

Hint: To select the Table without selecting any cells, drag across the Table or click on the Table handle.

2. Click the **Border** tab in the 'Format' dialog box.

3. In the 'Line' panel, select a line style from the 'Style' box, and select a line color from the 'Color' menu.

4. In the 'Border' panel, click the desired boundary to which you want to apply the current line style.

```
Note: To change the border color first set the color and then
select the border type.
```

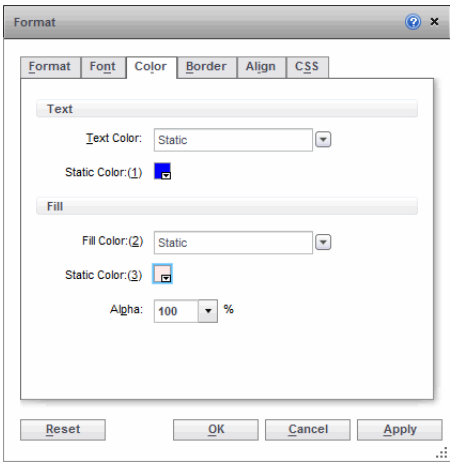
To set all borders to the current line style, click the 'All Border' button. To remove all borders or return to the default style, click the 'No Border' or 'Default' button, respectively.

See Also

Table Properties, for information on setting borders using Table styles.

2.3.5 Color Tab

The **Color** tab allows you to set text color, background color, and transparency for components.



Colors for Tables can also be set using Table Styles and highlighting.

For Tables, a text and background color can be set for the table as a whole, or independently for the title bar, columns headers, and column cells.

Setting Colors

To set a static foreground color, background color, and transparency (alpha) for a component, follow the steps below:

1. Right-click the element, and select ‘Format’ form the context menu. This opens the ‘Format’ dialog box.

   For a Table component, right-click on the region of the table to which you want to apply the property:

   a. Right-click a data cell to apply the property to all data cells in the column.

   b. Right-click on the column header cell to apply the property to only the column header.

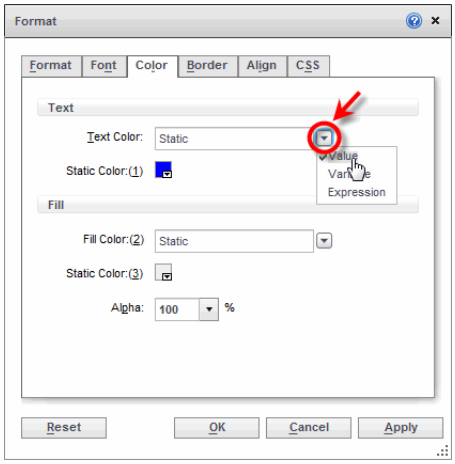
   c. Right-click the title bar to apply the property to only the title bar.

   d. Right-click the Table handle (with no cell selected) to apply the property to the whole Table. This global table color or transparency is overridden by the property specified for the individual cell.

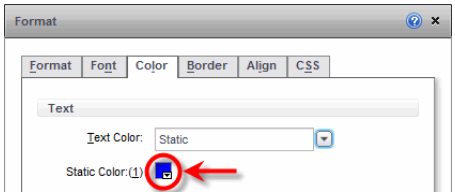
Hint: To select the Table without selecting any cells, drag across the Table or click on the Table handle.

2. Select the **Color** tab in the ‘Format’ dialog box.

3. In the ‘Text’ panel, select ‘Value’ from the right-side pop-up menu.

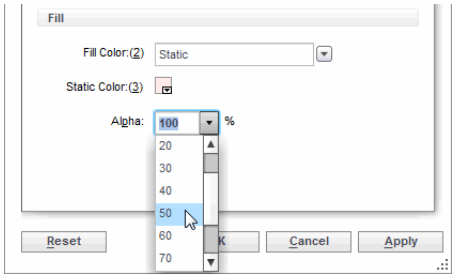


4. Click on the color chip to select the desired foreground (text) color.



5. Repeat the above steps in the ‘Fill’ panel to set the background (fill) color.

6. Set the fill transparency by selecting a value from the ‘Alpha’ menu. A value of 0% indicates complete fill transparency (i.e., fill color not visible), and a value of 100% indicates complete fill opacity.



You can also color dynamically by specifying an input control (variable) or an expression. See *Dynamic Properties* for details.

See Also

Adding Highlights to a Table, for information on conditional color assignments.

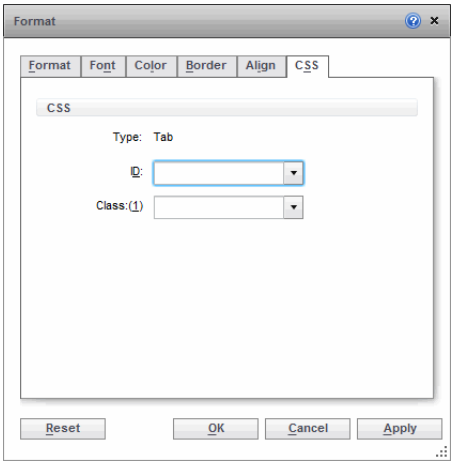
Table Properties, for information on setting borders using Table styles.

Dynamic Properties, for information on setting color via variable or expression.

2.3.6 CSS Tab

CSS is not available for *Shape Components* or for *Tables* that have a Table Style applied.

The **CSS** tab allows you to specify a CSS Class or ID to style the visual properties of a component. You can style individual regions of a component independently, just as with manual formatting. See *Format Dialog Box* for a list of selectable component regions.



There are two steps to using CSS properties to style components:

1. Creating the CSS definition file

2. Applying the CSS styles to Viewsheet components and component regions

The first step is generally performed by an administrator and requires access to Enterprise Manager. The following sections explain both steps and provide an example.

Creating the CSS Definition File

To prepare the CSS definition file, follow the steps below:

1. Create a CSS file (UTF-8 format, without BOM) that contains the desired Class, ID, or Type definitions. The formatting specified in the CSS file will be applied to all elements that are assigned the given Class or ID (see *Applying CSS Styles*), or that are of the specified Type.

    a. To specify a Class definition, use the following syntax:

    ```
    .HighlightColumn {
       background-color: yellow;
       ...
       ...
    }
    ```

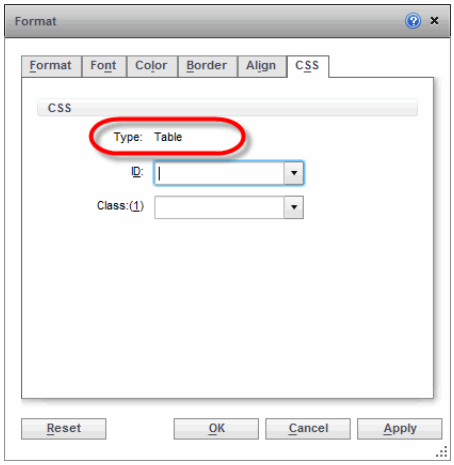
    b. To specify an ID definition, use the following syntax:

    ```
    #SalesTable {
       background-color: #FFEEBB;
       ...
       ...
    }
    ```

c. To specify a Type definition, use the following syntax:

```
Table Detail {
   background-color: #FFEEBB;
   ...
   ...
}
```

The Type definition applies styles universally to all element regions of a particular type (e.g., all table detail cells). You can find the 'Type' of an element region in the **CSS** tab for the selected element.



Formatting based on Type for a particular element region is superseded by formatting based on Class and ID (if specified) for that region.

2. Deploy the CSS file as described in the 'Viewsheet CSS' section of *Look-and-Feel* in *Advanced Server Administration*. (This may require an administrator.)

See *CSS Properties and Syntax* for details on which component properties can be assigned styles.

See Also
CSS Properties and Syntax, for details on which component properties can be assigned styles.

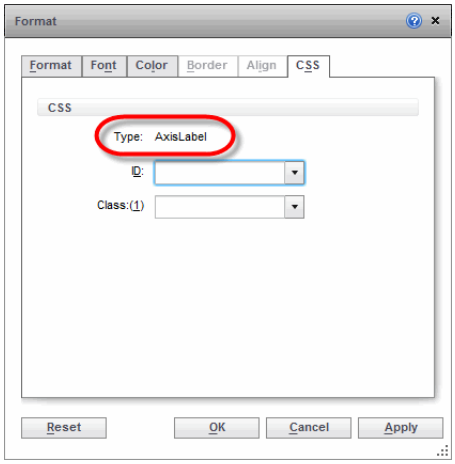
Applying CSS Styles

To set a component's visual properties by using CSS Class and ID, follow the steps below:

1. In Visual Composer, right-click on the region of the component you want to style, and select ‘Format’ from the context menu. This opens the ‘Format’ dialog box.

The **CSS** tab is not available for *Tables* that have a Table Style applied.

2. Select the **CSS** tab. The text next to the ‘Type’ label indicates the component region to which the CSS styles will be applied.



Note: CSS styles are overridden by manually-set styles. The ‘Reset’ button removes only the manually-set styles. Also, manually-set transparency will affect the appearance of CSS colors. See *Color Tab* for information on changing transparency.

3. On the **CSS** tab, select the CSS ‘ID’ and/or ‘Class’ to define the component’s style.

   You can select both an ‘ID’ and a ‘Class’ together, or select only one of these. Multiple components can share the same CSS Class name, which will then apply the same styles to these components. However, a CSS ID should be unique to the component.

The ‘Class’ and ‘ID’ menus list Classes and IDs that are defined in the deployed CSS definition file. (See *Creating the CSS Definition File*.) You can also type in a new Class or ID name, but this will not apply any styles until a corresponding definition is created in the CSS definition file.

Example: Creating and Applying CSS Styles

In this example, you will first create a new CSS file, and then design a Viewsheet that uses the styles defined in the file. Follow the steps below.

1. Create a plain text file with the following contents.

```
.Header2 {
   font-size: 20pt;
   font-family: "Arial Narrow";
   font-weight: bold;
```

```
   background-color: #FFEEBB;
   color: rgb(100,50,50);
   text-align: center;
   vertical-align: middle;
   border-color: rgb(150,50,50);
   border-style: double dotted;
}

#SalesTable {
   background-color: #FFEEBB;
}

.HighlightColumn {
   background-color: yellow;
}

.AxisText {
   color: rgb(150,50,50);
   font-size: 14pt;
   font-weight: bold;
}
```

2. Save the text file in UTF-8 format (without BOM) with the name '`myStyle.css`'.

3. Deploy the CSS file by following the instructions in the 'Viewsheet CSS' section of *Look-and-Feel*, in *Advanced Server Administration*. (This may require an administrator.)

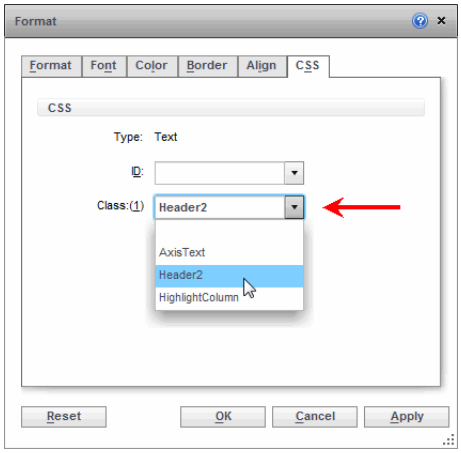
This CSS file is now ready for use. It contains three CSS Classes (Header2, HighlightColumn, and AxisText) and one CSS ID (SalesTable). In the next steps, you will create a new Viewsheet with components that draw their styles from these CSS classes.

4. Create a new Viewsheet based on the 'Tutorial' > 'ProductSales' Worksheet.

5. Add a Text element to the Viewsheet. Enter the text 'Total Sales'.

6. Right-click the Text component, and select 'Format' from the context menu. This opens the 'Format' dialog box.

7. Click the **CSS** tab. From the 'Class' menu, select 'Header2', and click 'OK'. This applies the Header2 style to the text element.



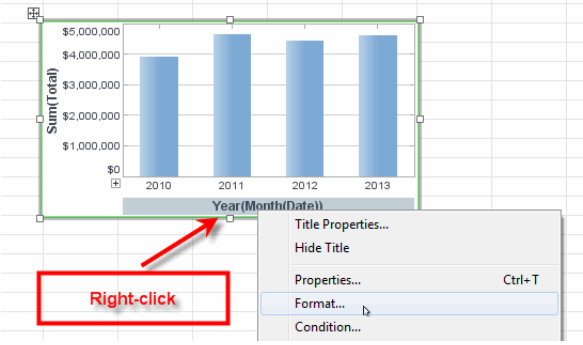
8. Resize the text element so that the text is fully visible.



9. Add a new Chart component. From the **Data Source** panel, drag the 'Date' field to the 'X' region, and drag the 'Total' field to the 'Y' region.

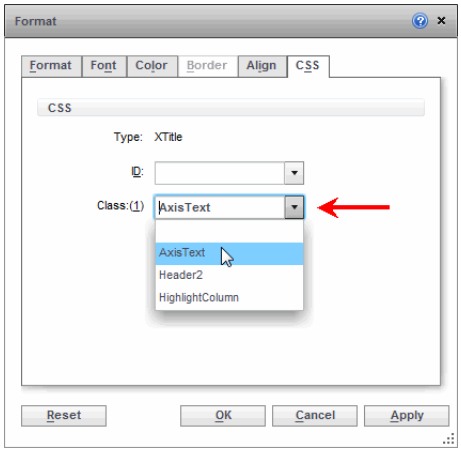
10. Click 'Select Chart Style' in the **Data** panel, and select the bar-chart option. This changes the display to a bar chart.

11. Right-click the X-axis title on the chart, and select 'Title Format' from the context menu. This opens the 'Title Format' dialog box.



Note that the 'Type' field indicates 'XTitle'. This means that the CSS only applies to the X-axis title.

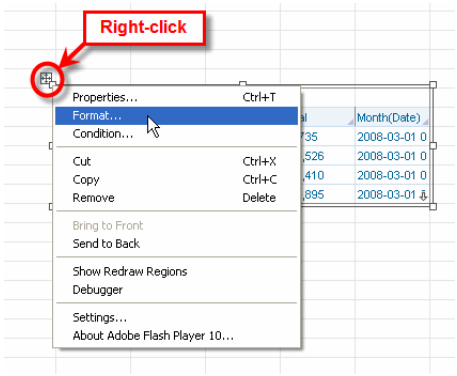
12. In the 'CSS' region, select 'AxisText' from the 'Class' menu. Click 'OK'. This applies the AxisText style to the X-axis title.



13. Repeat the above steps to apply the AxisText style to the Y-axis title as well, and then close the Chart Editor.

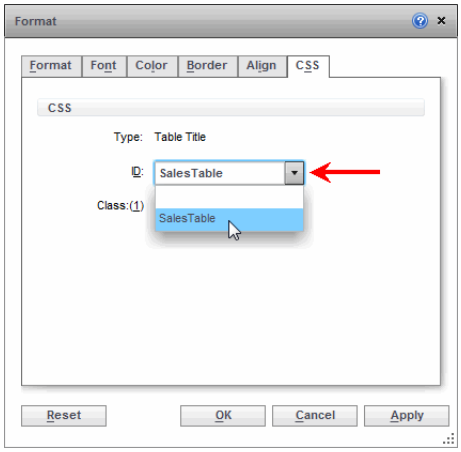
14. From the Component tree, drag the entire ‘SalesByDate’ Data Block to the Viewsheet. This creates a new table called ‘SalesByDate’.

15. Click to select the table. Click a second time on the table’s “move” handle to ensure that the entire table is selected (rather than just the title bar). Finally, right-click the table’s “move” handle, and select ‘Format’ from the context menu. This opens the ‘Format’ dialog box for the table as a whole.

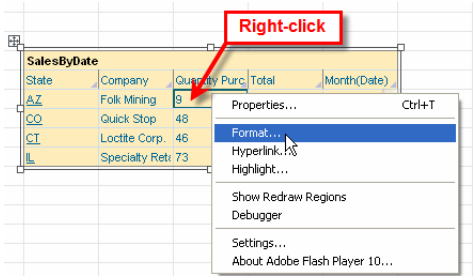


Note that the ‘Type’ field indicates ‘Table’. This means that the CSS applies to the table as a whole

16. Click the **CSS** tab. From the ‘ID’ menu, select ‘SalesTable’, and click ‘OK’. This applies the SalesTable style to the table as a whole.

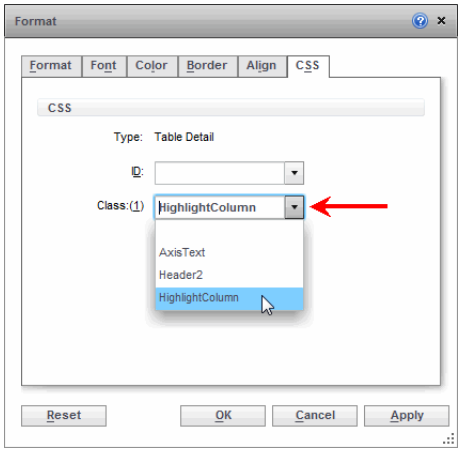


17. Click to select a cell in the 'Quantity Purchased' column of the table. Right-click the cell, and choose 'Format' from the context menu. This opens the 'Format' dialog box for the data cells in the column.



Note that the 'Type' field indicates 'Table Summary'. This means that the CSS only applies to the data cells

18. Click the **CSS** tab. From the 'Class' menu, select 'HighlightColumn', and click 'OK'. This applies the HighlightColumn style to the data cells in the 'Quantity Purchased' column.



You can manually make additional modifications to the chart, as desired. (For example, change the color of the chart bar elements to match the axis titles.) Format settings that you adjust manually will override format settings controlled by CSS.



SalesByDate

State	Company	Quantity Purc	Total	Month(Date)
AZ	Folk Mining	9	$6,735	2008-03-01 0
CO	Quick Stop	48	$10,526	2008-03-01 0
CT	Loctite Corp.	46	$16,410	2008-03-01 0
IL	Specialty Ret	73	$19,895	2008-03-01 0

¢

See Also

Alignment Tab, *Font Tab*, *Border Tab*, *Color Tab*, to set styles without using CSS.

Dynamic Properties, for information on setting color via variable or expression.

CSS Properties and Syntax

The following attributes can be styled by CSS:

– Alignment
– Font
– Border
– Color

The table below lists the available properties and values.

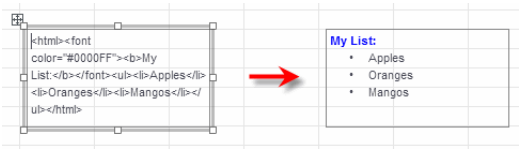
PROPERTY	AVAILABLE VALUES
text-align	‘left’, ‘right’, ‘center’
vertical-align	‘top’, ‘middle’, ‘bottom’
word-wrap	‘normal’, ‘break-word’
font-style	‘normal’, ‘italic’
font-weight	‘normal’, ‘bold’
font-size	‘xx-small’, ‘x-small’, ‘small’, ‘medium’, ‘large’, ‘x-large’, ‘xx-large’, or size in pixels
font-family	Font name
font	Shortcut: font-style font-weight font-size font-family
text-decoration	‘none’, ‘underline’, ‘line-through’

PROPERTY	AVAILABLE VALUES
color	Color value (hex, rgb, color string)
background-color	Color value (hex, rgb, color string) or ‘transparent’
opacity	Float in range [0, 1]
border-style	Line style: ‘none’, ‘dotted’, ‘dashed’, ‘solid’, ‘double’ Single value applies to all borders
border-top-style border-right-style border-bottom-style border-left-style	Line style: ‘none’, ‘dotted’, ‘dashed’, ‘solid’, ‘double’
border-color	Color value (hex, rgb, or color string) Single value applies to all borders, or Four-values (top right bottom left), or Two values (top/bottom right/left)

2.4 Using HTML in Viewsheet Text

You can display HTML content within any element that displays text (e.g., Table cell, Text component, Selection List, etc.). To do this, surround the HTML markup with `<html>...</html>` tags. For example, the following markup placed within a Text element creates a list within the Text element:

```
<html><font color="#0000FF"><b>My List:</b></
font><ul><li>Apples</li><li>Oranges</li><li>Mangos</li></
ul></html>
```



The HTML tags supported are the same as those supported by the Flex label.htmlText component: `<a>`, `<b>`, `<br>`, `<font>`, `<img>`, `<i>`, `<li>`, `<p>`, `<span>`, `<textformat>`, and `<u>`.

3 Data View Components

A Viewsheet typically contains at least one data view that presents the primary analysis result. *Data views* can be in the form of a simple table, a crosstab (pivot table), a chart, or a geographical map.

All data view elements are listed under the 'Toolbox > Data View' node of the Component tree.

See Also
Output Components, for information on graphical presentation of data.
Selection Components, for information on filtering the data view display.
Input Components, for information on dynamically modifying Worksheet results.

3.1 Tables

This section explains how to create Viewsheet tables and set their properties.

3.1.1 Creating a Table

There are several ways to create a new table.

Creating a Table by Dragging a Data Block

To create a table follow these steps:

Dragging individual Data Block columns will create a Selection List/Tree, not a Table.

1. Select a complete Data Block in the Component tree.
2. Drag the entire Data Block directly onto the Viewsheet grid.

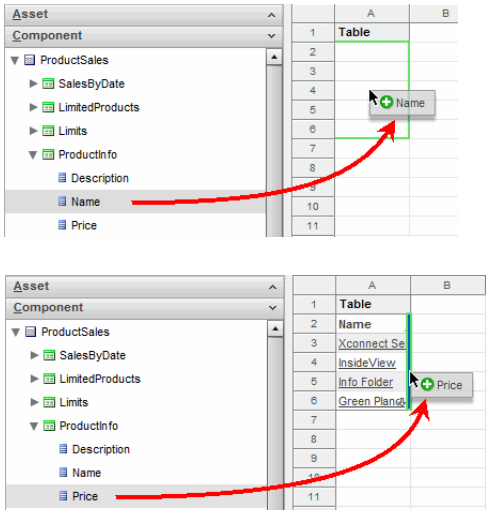
Creating a Table with the Table Element

To create a table follow these steps:

1. Drag a Table element from the Component tree into the Viewsheet grid. This creates an empty table element.
2. Expand a Data Block node in the Component tree, and drag and drop one or more columns onto the Table element.

```
Note: Cube dimensions and measures cannot be added to a Table.
```

3. Repeat to add additional columns. A green highlight on the Table indicates where the dragged columns will be placed.



See Also

Creating an Embedded Table, for information on displaying an Embedded Table.

Creating a Crosstab, for information on generating pivot tables.

3.1.2 Creating an Embedded Table

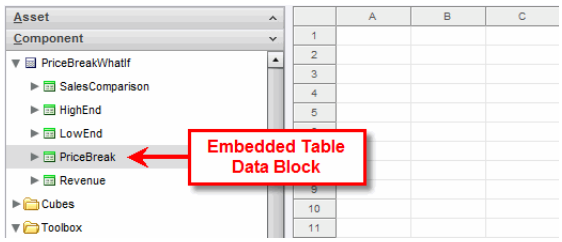
An Embedded Table is a table into which you can manually enter data. Embedded tables can be edited both within Visual Composer and within the Portal, allowing the end-user to enter data in tabular form.

`Note: Selection elements do not affect Embedded Tables.`

An Embedded Table in a Viewsheet is always connected to an Embedded Table in the underlying Worksheet. Edits that you make to the Viewsheet's Embedded Table are automatically propagated to the associated Worksheet table at runtime. Data that you enter is saved with the Viewsheet, not with the Worksheet, so edits that you make to one Viewsheet do not affect other Viewsheets based on the same Worksheet.

To add a full Embedded Table Data Block, follow the steps below:

1. Select an Embedded Table Data Block from the data source in the **Component** panel.



2. Drag the entire Data Block directly onto the Viewsheet grid. This adds the entire Data Block to the Viewsheet as an Embedded Table.

Ctrl-click to select multiple cells.

3. (Optional) To delete an unwanted column or columns, click to select the unwanted column header cell. Right-click on the selected cell(s), and choose 'Delete Column(s)' from the context menu.

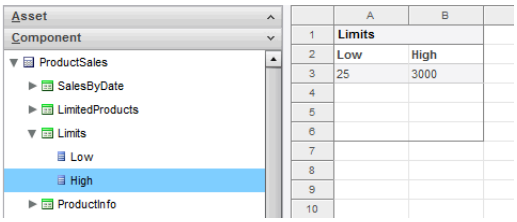
To build an Embedded Table column-by-column, follow the steps below:

1. Drag a Table element from the Component tree into the Viewsheet grid. This creates an empty Table element.

2. Expand an Embedded Table Data Block in the **Component** panel.

3. Drag-and-drop an individual column or columns onto the empty Table element.

4. Repeat to add additional columns. A green highlight on the Table indicates where the dragged columns will be placed.



See Also

Adding Hyperlinks to an Embedded Table, for information on using hyperlinks.
Table Properties, for Embedded Table properties (same as regular table).
Input Components, for providing user input via graphical controls.
Format Dialog Box, for the available table formatting options.
Creating an Embedded Table, in *Advanced Data Mashup*.
Accessing Component Data, in *Advanced Dashboard Scripting*, to set properties using embedded data.

3.1.3 Table Properties

The Table component's 'Properties' dialog box provides **General**, **Advanced**, and **Script** tabs. The next sections discuss the component-specific properties available under these tabs.

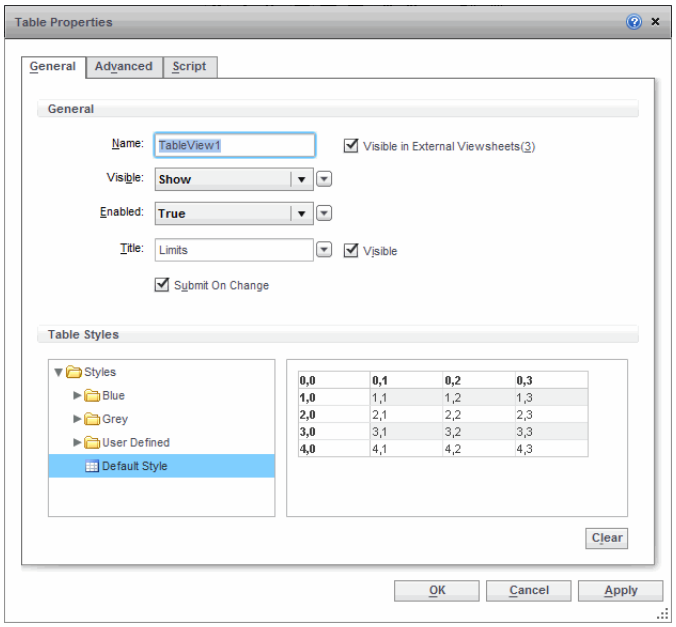
See Also

Crosstab Properties, for information about Crosstab properties.
Properties Dialog Box, for information about common component properties.
Format Dialog Box, for information on the 'Format' dialog box.

Table Properties: General Tab

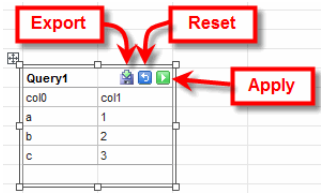
In addition to the common properties, the **General** tab in the 'Table Properties' dialog box allows you to select a layout theme from the 'Table Styles' panel.

A table style is a set of predefined formats (fonts, alignments, backgrounds etc.) that is directly applied to any table. The 'Table Styles' panel provides access to a Style Library containing a large number of predefined table styles. To select a table style, expand the desired category node (e.g. 'Grid', 'Columns', 'Rows', etc.) and click on one of the styles. A preview is of the style is displayed on the right side.



The **General** tab of the 'Properties' dialog box for an Embedded Table additionally provides a 'Submit On Change' option. When this option is enabled (default), any changes that you make to entries in the Embedded Table are immediately submitted to the linked Embedded Table in the underlying Worksheet, which immediately updates any dependent data views in the Viewsheet.

When 'Submit On Change' is disabled, changes that you make to entries in the Embedded Table are not submitted until you press the 'Apply' button in the table title bar. Click the 'Reset' button to restore the table's entries to their most recently submitted state.



See Also

General Tab, for information on properties common to the **General** tab.
Border Tab, for information on manually setting border styles.
Color Tab, for information on manually setting text and background colors.
Creating a Custom Table Style, in *Advanced Report Design*, to create new table styles.

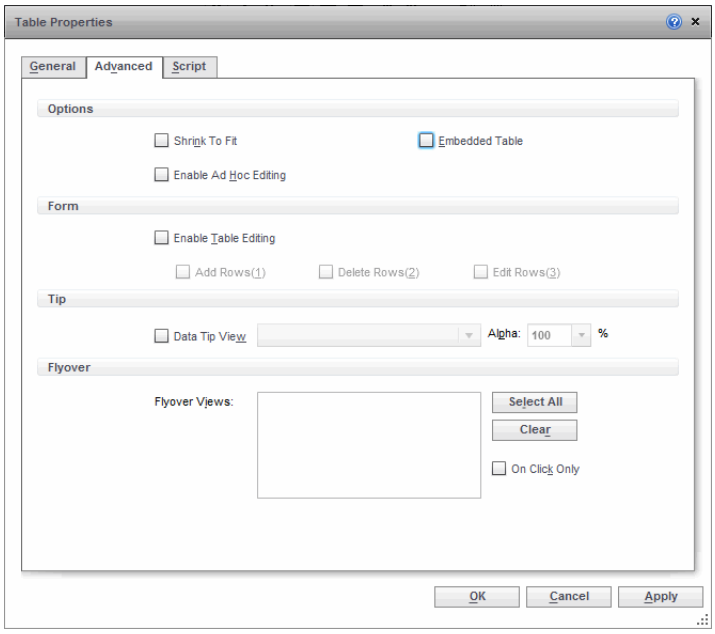
Table Properties: Advanced Tab

The **Advanced** tab in the 'Table Properties' dialog box provides the following Table-specific properties:

Shrink to Fit	When enabled, this reduces the height of the table to match the number of records displayed. For example, if the original table height (as designed) was 10 rows, and the user then makes selections which restrict the result set to five rows, the table's bottom border will retract or "shrink" to eliminate any empty rows at the bottom of the table.
Embedded Table	When enabled, specifies that the table is an Embedded Table. This makes the table cells editable by the user and removes any hyperlinks. (This option is only available if the table was originally created as an Embedded Table. See *Creating an Embedded Table* for more information.)
Enable Table Editing	This option allows users to modify various aspects of a table by inserting, removing, or changing data records. To perform validation checks on data entered by users, see *Setting Column Editing Options*. Modified table data can be written back to the database by an appropriate Viewsheet script. See *Accessing User-Modified Data in a Table* in *Advanced Dashboard Scripting* for information. See *Editing a Crosstab* in *Interacting and Collaborating* for information on modifying a table in the Portal. The 'Add Rows', 'Delete Rows', and 'Edit Rows' settings are discussed below.
Add Rows	Allows a user to add new rows to the table. To add a new row, the user should right-click a row in the table and select 'Insert Row' or 'Append Row' from the context menu to respectively add a row before or after the clicked row. Added rows are not retained across sessions unless they are committed to the database by a Viewsheet script (see above). See *Setting Column Editing Options* for information about validation.
Delete Rows	Allows a user to delete rows from the table. To do this, the user should right-click a row in the table and select 'Delete Row' from the context menu. Deletions are not retained across sessions unless they are committed to the database by a Viewsheet script (see above).

Shrink to Fit	When enabled, this reduces the height of the table to match the number of records displayed. For example, if the original table height (as designed) was 10 rows, and the user then makes selections which restrict the result set to five rows, the table's bottom border will retract or "shrink" to eliminate any empty rows at the bottom of the table.
Edit Rows	Allows a user to edit data in the table. Edited data is not retained across sessions unless committed to the database by a Viewsheet script (see above). See *Setting Column Editing Options* for information about validation.
Data Tip View	A Viewsheet component is displayed when mouse hovers over table area. The component is filtered based on the hover region. See *Data Tip View* for more details.
Flyover Views	Allows you to select a set of Viewsheet components which will be adaptively filtered as the user hovers the mouse or clicks on the table area. (Select 'On Click Only' if you want the filtered components to update when the user clicks.) If the table contains aggregated data, filtering is based on the grouping fields (dimensions) in the row on which the user hovers or clicks. If the table contains detail data, filtering is based on the non-numeric columns in the row on which the user hovers or clicks. See *Flyover View* for more details.

`Note: End-user table editing options are available only if product is licensed for Viewsheet Forms.`



See Also

Accessing User-Modified Data in a Table, in *Advanced Dashboard Scripting*, for information on reading and committing modified data.

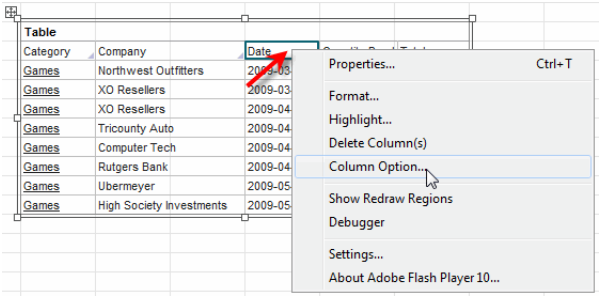
3.1.4 Setting Column Editing Options

Note: Column editing options are available only if product is licensed for Viewsheet Forms.

When you configure a table as “editable” (see *Table Properties: Advanced Tab*), you can place restrictions on which columns may be edited by users, and what types of values can be entered.

To set the editing options for a particular column, follow the steps below:

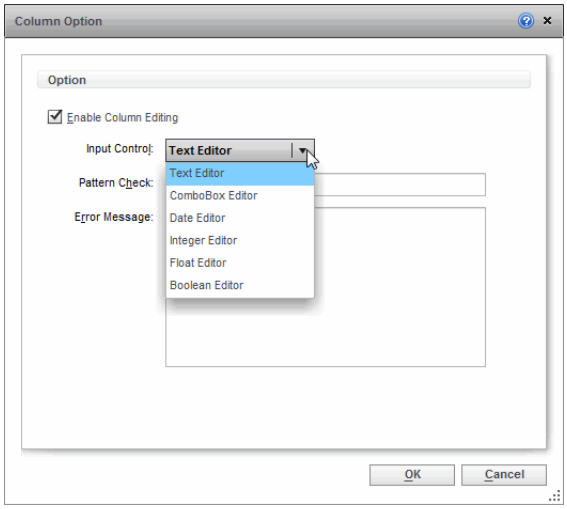
1. Right-click the column header, and select ‘Column Option’ from the context menu. This opens the ‘Column Option’ dialog box.



2. In the ‘Column Option’ dialog box, select ‘Enable Column Editing’ to make the column editable by users.

Note: The Viewsheet’s *onRefresh Handler* script is re-executed each time the user makes an edit to an editable table. See *Accessing User-Modified Data in a Table*, in *Advanced Dashboard Scripting*, for more details.

3. In the ‘Column Editor’ menu, select the type of editing control you wish to provide to users. The available options are ‘Text Editor’, ‘ComboBox Editor’, ‘Date Editor’, ‘Number Editor’, ‘Float Editor’, and ‘Boolean Editor’.



4. Press ‘OK’ to close the dialog box.

Each type of column editor provides settings which allow you to restrict the values that a user may enter into the column. This allows you to perform validation checking on user input. See *Validating User Input* for the available settings.

You can apply formatting to a editable column in the usual manner (see *Format Dialog Box*). Formatting is applied to existing column data as well as to any data that a user adds or edits in the column.

See Also
Table Properties: Advanced Tab, for information on enabling end-user table editing.

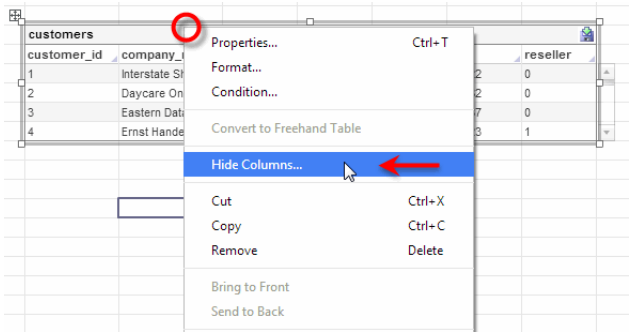
3.1.5 Hiding Table Columns

```
Note: Column hiding options are available only if product is
licensed for Viewsheet Forms.
```

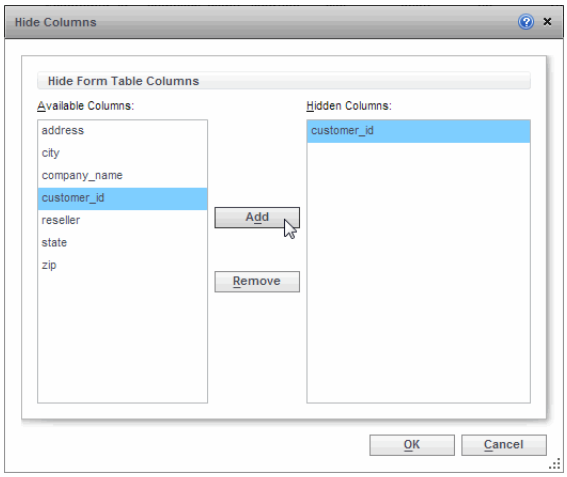
When you configure a table as “editable” (see *Table Properties: Advanced Tab*), you can choose to hide specific table columns. This is useful in cases when the table includes a primary key column (used to write data back to the database) that should not be displayed to the user.

To set a column of an editable table as a hidden column, follow the steps below:

1. Right-click the table title bar, and select ‘Hide Columns’ from the context menu. This opens the ‘Hide Column’ dialog box.



2. In the ‘Available Columns’ panel, select the columns you want to hide (Ctrl-click or Shift-click to select multiple columns), and press ‘Add’.



3. Press ‘OK’ to close the dialog box and hide the columns.

To retrieve the value of a hidden column, use the script function *getHiddenColumnValue(col,row)*. See *Accessing User-Modified Data in a Table*, in *Advanced Dashboard Scripting*, for more details.

See Also
Table Properties: Advanced Tab, for information on enabling end-user table editing.

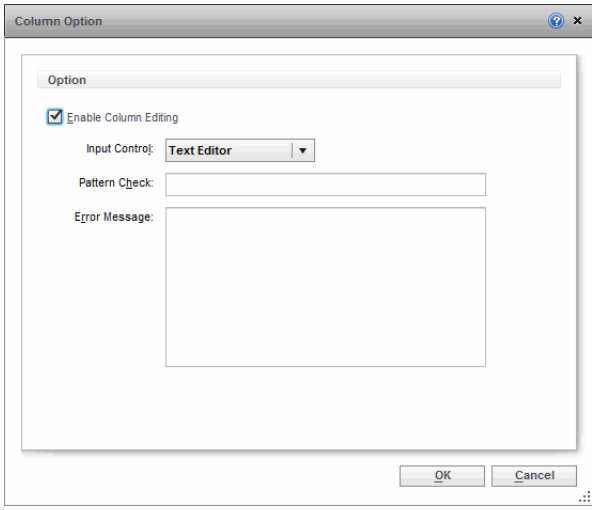
3.1.6 Validating User Input

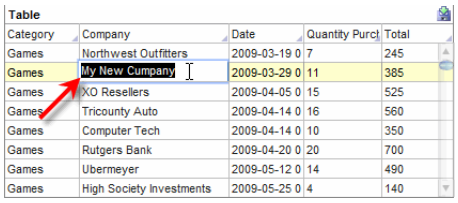
You can configure *TextInput* components and editable table columns with different types of input controls and validation features. The following sections describe the available editors for *TextInput* components and editable table columns.

See Also
Table Properties: Advanced Tab, for information on enabling end-user table editing.

Text Editor

The ‘Text Editor’ provides the user with standard text field input. This allows a user to simply type values into an editable table cell or *TextInput* component. To mask the entered text, use the ‘Password Editor’ option instead of the ‘Text Editor’ option.





The following options are available in the dialog box:

Pattern Check	Specifies a regular expression (regex) which the input must match. Failure to match the regular expression will cause the error message to be displayed.
Error Message	A text message to display when the pattern check fails. The placeholder “`{0}`” can be used to insert the user-entered value at a desired location in the error message. Example error message: “`The value you entered, {0}, is not acceptable.`”

You can find many regex tutorials and examples on the Internet (e.g., as http://www.regular-expressions.info). Some sample expressions and error message combinations are show below:

PATTERN CHECK	ERROR MESSAGE (SUGGESTED)
^[a-zA-Z0-9_]*$	Input can contain only numbers, letters, and underscore (_).
\d{3}-\d{3}-\d{4}	Input must be a phone number with form ###-###-####.
^[a-zA-Z0-9_]{0,15}$	Input length must be 15 characters or fewer.

The following is a summarization of the regular expression syntax.

```
Alternatives separated by |
Quantified atoms
          {n,m} Match at least n but not more than m times.
   {n,} Match at least n times.
   {n} Match exactly n times.
   * Match 0 or more times.
   + Match 1 or more times.
   ? Match 0 or 1 times.
a . matches everything except \n
a ^ is a null token matching the beginning of a string or
line (i.e., the position right after a newline or right
before the beginning of a string)
a $ is a null token matching the end of a string or line
(i.e., the position right before a newline or right after
the end of a string)
Character classes (e.g., [abcd]) and ranges (e.g., [a-z])
Special backslashed characters work within a character class
(except for backreferences and boundaries).
\b is backspace inside a character class
Special backslashed characters
\b null token matching a word boundary (\w on one side and
\W on the other)
\B null token matching a boundary that isn't a word boundary

\A Match only at beginning of string
\Z Match only at end of string (or before newline at the
end)
\n newline
\r carriage return
\t tab
\f formfeed
\d digit [0-9]
\D non-digit [^0-9]
\w word character [0-9a-z_A-Z]
\W a non-word character [^0-9a-z_A-Z]
\s a whitespace character [ \t\n\r\f]
\S a non-whitespace character [^ \t\n\r\f]
\xnn hexadecimal representation of character
\cD matches the corresponding control character
\nn or \nnn octal representation of character unless a
backreference.
a \1, \2, \3, etc. match whatever the first, second, third,
etc. parenthesized group matched. This is called a
backreference. If there is no corresponding group, the
number is interpreted as an octal representation of a
character.
\0 matches null character
Any other backslashed character matches itself
Expressions within parentheses are matched as subpattern
groups and saved for use by certain methods.
```

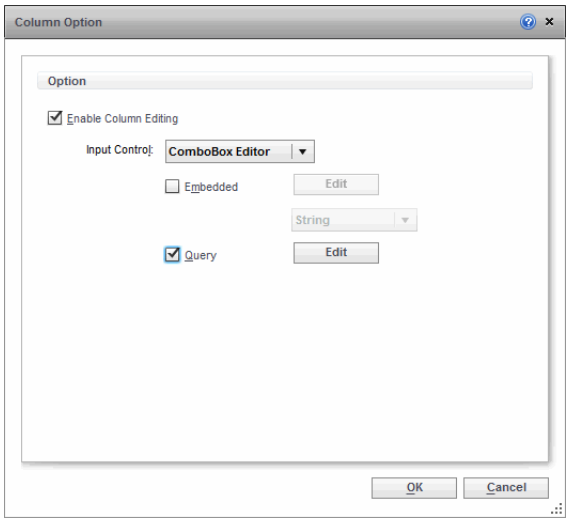
See Also

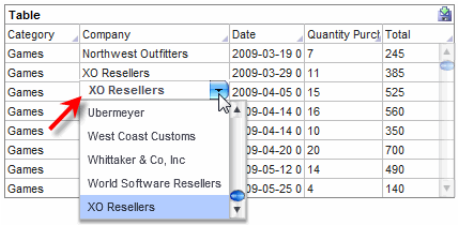
Setting Column Editing Options, for information on setting column options.

Table Properties: Advanced Tab, for information on enabling end-user table editing.

ComboBox Editor

The ‘ComboBox Editor’ provides the user with standard menu input. This allows a user to simply select a value from a predefined list.





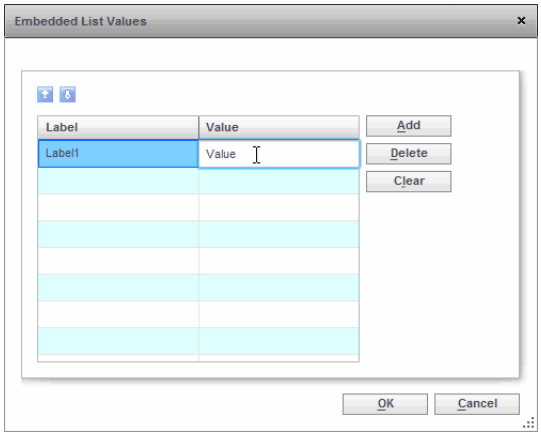
To populate the menu items from a manually created list, follow the steps below:

1. In the ‘Column Option’ dialog box, select the ‘Embedded’ option and click on the ‘Edit’ button. This opens the ‘Embedded List Values’ dialog box.

2. Click the ‘Add’ button. This will populate the list with blank ‘Label’ and ‘Value’ fields.

3. Double-click on the list elements, and enter a desired ‘Label’ and ‘Value’.

4. Repeat the above steps to insert more elements.

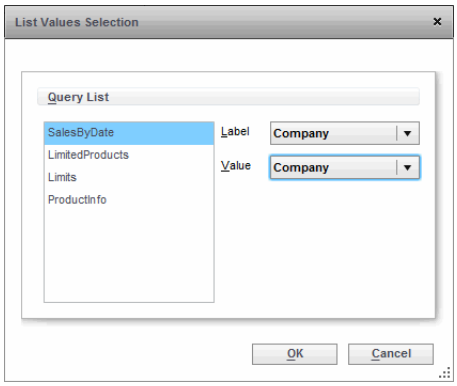


5. Click ‘OK’ to close the dialog box.

To populate the menu items from the Viewsheet’s data source, follow the steps below:

1. In the ‘Column Option’ dialog box, select the ‘Query’ option, and click on the ‘Edit’ button. This opens the ‘List Values Selection’ dialog box.

2. Select a Data Block from the list on the left side. Select the appropriate fields from the ‘Label’ and ‘Value’ menus.



3. Click ‘OK’ to close the dialog box.

Enable both the ‘Embedded’ and ‘Query’ options to merge together the values retrieved from both sources into a single list.

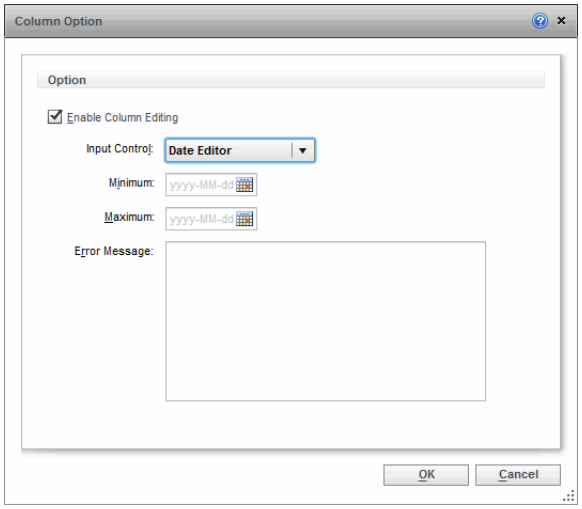
See Also

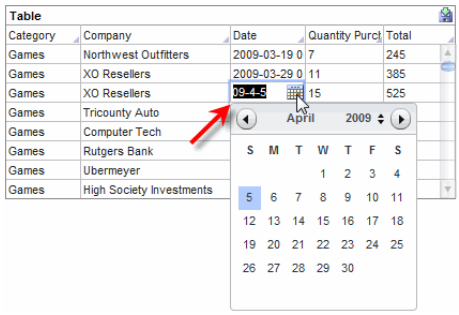
Setting Column Editing Options, for information on setting column options.

Table Properties: Advanced Tab, for information on enabling end-user table editing.

Date Editor

The ‘Date Editor’ provides the user with a pop-up calendar on which to select a date. This allows a user to enter date values into an editable table cell or *TextInput* component.





The following options are available in the dialog box:

Minimum	Specifies the start date for the permitted time interval. If the user enters a date earlier than the minimum, the error message will be displayed.
Maximum	Specifies the end date for the permitted time interval. If the user enters a date later than the maximum, the error message will be displayed.
Error Message	A text message to display when the user enters a date outside the specified interval. The placeholder “`{0}`” can be used to insert the user-entered value at a desired location in the error message. Example error message: “`The value you entered, {0}, is not acceptable.`”

See Also

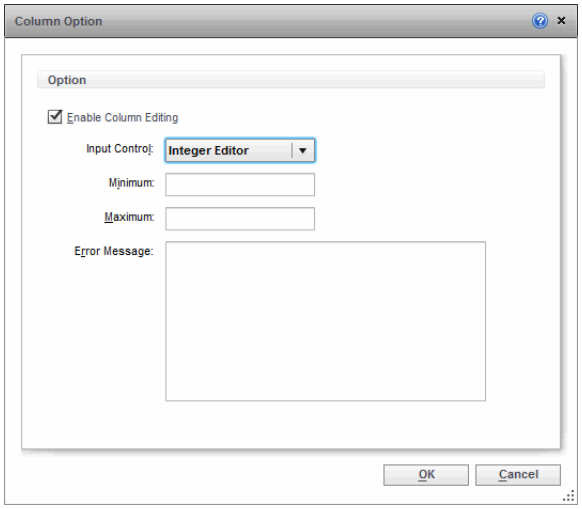
Setting Column Editing Options, for information on setting column options.

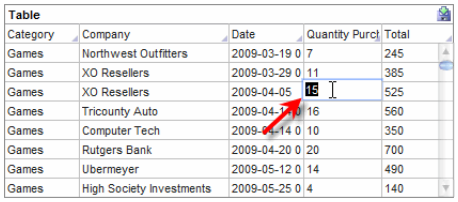
Table Properties: Advanced Tab, for information on enabling end-user table editing.

Integer Editor

Alphabetical characters and decimal points are not permitted. To enter decimals, use the *Float Editor*.

The ‘Integer Editor’ provides the user with standard integer field input. This allows a user to type integer values into an editable table cell or *TextInput* component.





The following options are available in the dialog box:

Minimum	Specifies the lowest allowed integer value. If the user enters a value lower than the minimum, the error message will be displayed.
Maximum	Specifies the highest allowed integer value. If the user enters a value higher than the maximum, the error message will be displayed.
Error Message	A text message to display when the user enters a value outside the specified interval. The placeholder “`{0}`” can be used to insert the user-entered value at a desired location in the error message. Example error message: “`The value you entered, {0}, is not acceptable.`”

See Also

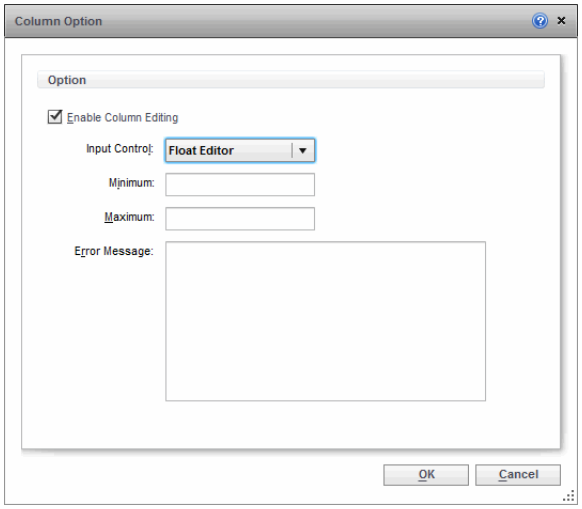
Setting Column Editing Options, for information on setting column options.

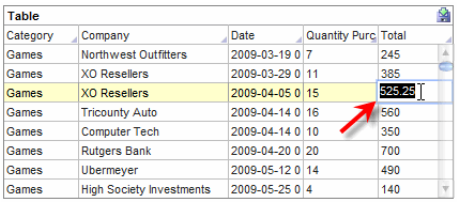
Table Properties: Advanced Tab, for information on enabling end-user table editing.

Float Editor

Alphabetical characters are not permitted.

The ‘Float Editor’ provides the user with a standard floating point field input. This allows a user to type decimal values into an editable table cell or *TextInput* component.





The following options are available in the dialog box:

Minimum	Specifies the lowest allowed floating point value. If the user enters a value lower than the minimum, the error message will be displayed.
Maximum	Specifies the highest allowed floating point value. If the user enters a value higher than the maximum, the error message will be displayed.
Error Message	A text message to display when the user enters a value outside the specified interval. The placeholder “`{0}`” can be used to insert the user-entered value at a desired location in the error message. Example error message: “`The value you entered, {0}, is not acceptable.`”

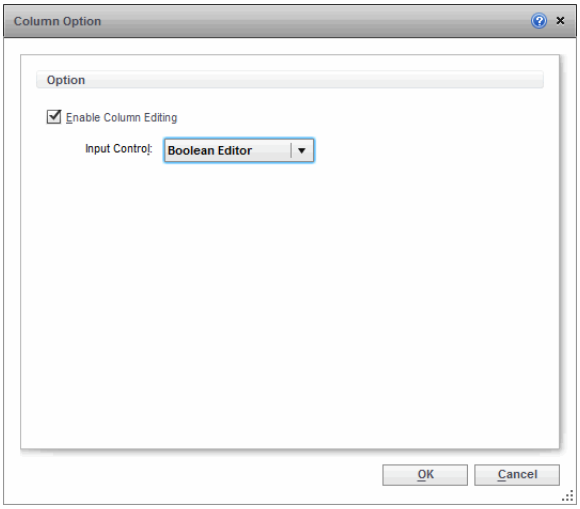
See Also

Setting Column Editing Options, for information on setting column options.

Table Properties: Advanced Tab, for information on enabling end-user table editing.

Boolean Editor

The 'Boolean Editor' provides the user with a simple checkbox input. This allows the user to enter "true" (checked) or "false" (unchecked) into an editable table cell or *TextInput* component.



See Also

Setting Column Editing Options, for information on setting column options.

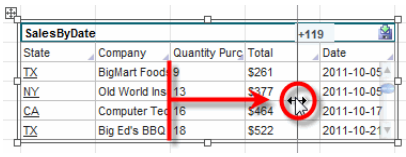
Table Properties: Advanced Tab, for information on enabling end-user table editing.

Password Editor

The 'Password Editor' option is the same as the *Text Editor* option, but masks the entered text using '*' characters.

3.1.7 Resizing a Table Column

To resize a table column, click and drag on the column border to obtain the desired width.



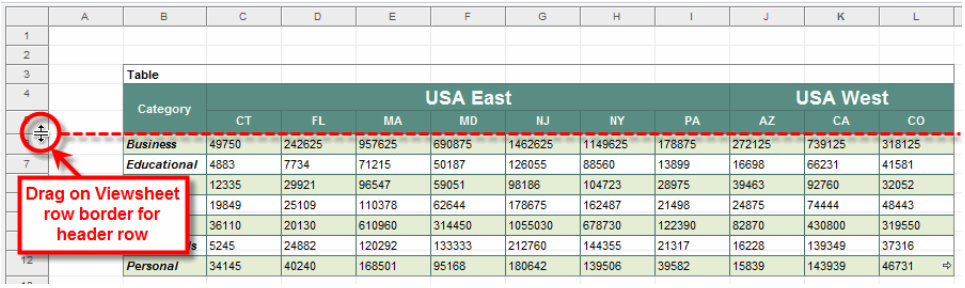
See Also

Resizing the Viewsheet Grid, for information on changing the global grid spacing.

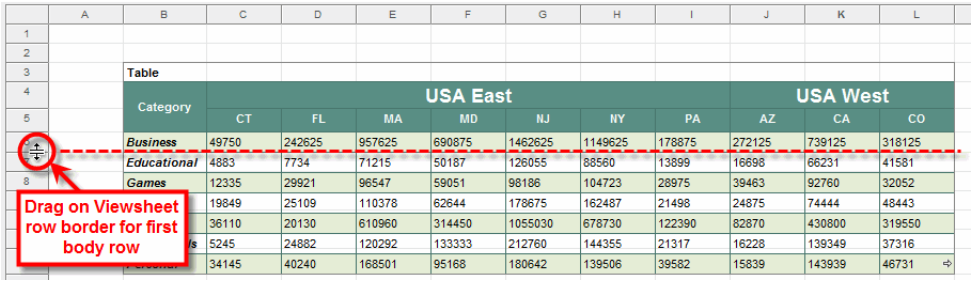
3.1.8 Resizing a Table Row

To resize a table header row, click and drag on the Viewsheet row border corresponding to the desired header row. Header rows can be resized

independently of body rows. For a Freehand Table, the number of header rows is defined in the *Freehand Table Properties: Advanced Tab*.



To resize the table body rows, click and drag on the Viewsheet row border corresponding to the *first body row*. All other body rows adopt the same size as the first body row.



See Also

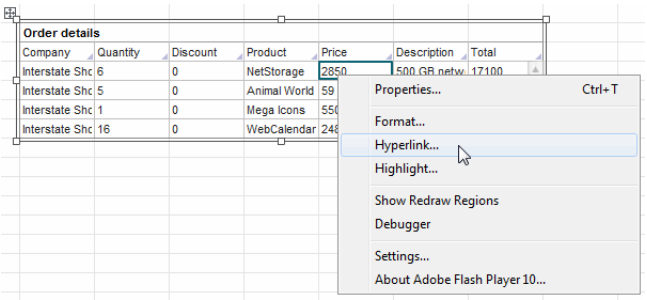
Resizing the Viewsheet Grid, for information on changing the global grid spacing.

3.1.9 Adding Hyperlinks to a Table

In tables, you can attach hyperlinks to a specific columns.

Adding Hyperlinks to a Regular Table

To add hyperlinks to regular table data, right-click the table column to which you wish to add the links, and select 'Hyperlink' from the context menu.



This opens the ‘Hyperlink’ dialog box. See the *Hyperlinks* section for complete instructions on how to use the ‘Hyperlink’ dialog box.

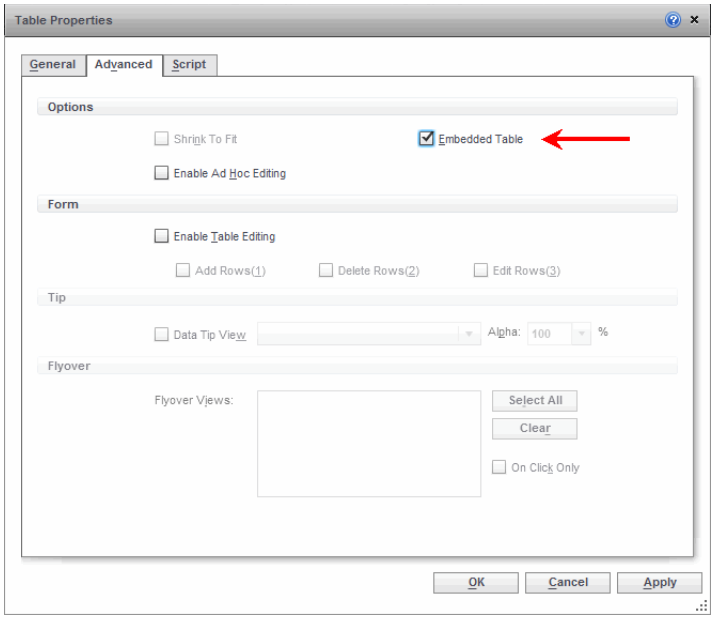
Adding Hyperlinks to an Embedded Table

It is not possible to attach hyperlinks to the cells of an Embedded Table. However, you can convert the Embedded Table into a regular table, and then add the hyperlinks to the regular table.

To add hyperlinks to an Embedded Table using this method, follow the steps below:

1. Right-click on the Embedded Table, and select ‘Properties’ from the context menu. This opens the ‘Table Properties’ dialog box.

2. Disable the ‘Embedded Table’ option, and press ‘OK’.



3. Add hyperlinks to the desired column(s) as described in *Hyperlinks*.

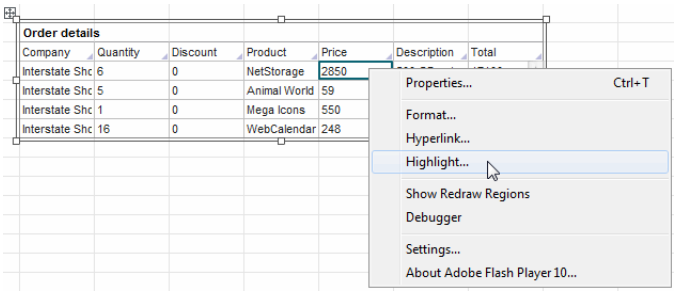
To restore the editability of the table, re-enable the ‘Embedded Table’ option in the ‘Table Properties’ dialog box. Note: This will remove any hyperlinks from the table.

See Also

Creating an Embedded Table, for information on how to create an embedded table.

3.1.10 Adding Highlights to a Table

To add highlights to table data, right-click the table column to which you wish to add the highlight, and select ‘Highlight’ from the context menu.



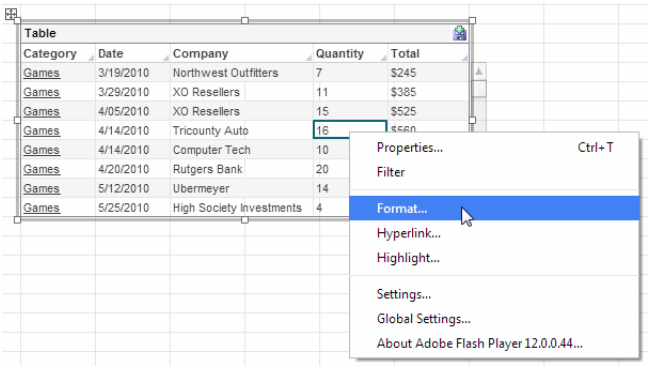
This opens the ‘Highlight’ dialog box. See the *Highlights* section for complete instructions on how to use the ‘Highlight’ dialog box.

3.1.11 Displaying Table Cell Values Graphically

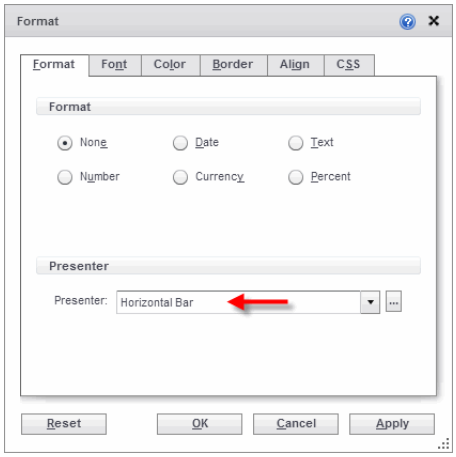
To display cell values using a graphical representation such as a colored bar or QR code, follow the steps below.

1. Select the table cell that you want to display graphically.

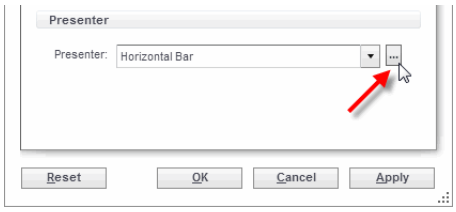
2. Right-click on the cell, and select ‘Format’ from the context menu.



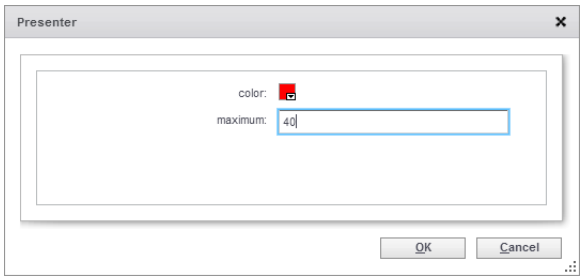
3. On the **Format** tab of the ‘Format’ dialog box, select a desired graphical representation from the ‘Presenter’ menu.



4. Press the ‘Options’ button to open the ‘Presenter’ dialog box.

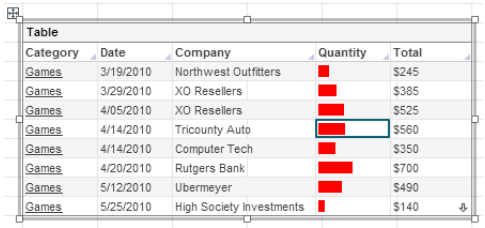


5. Specify the desired values for the presenter options, and press ‘OK’.



See *Presenter* for more information about the options.

6. Press ‘OK’ to close the ‘Format’ dialog box. This applies the presenter to the selected cells.



See Also

Displaying Data in a Text Component Graphically, to use a presenter in a Text component.

Presenter, for more information about presenters.

3.1.12 Changing Table Column Headers

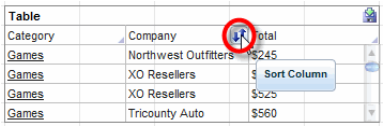
You can also double-click the column header to modify the text.

To change or format table column headers, right-click the desired column header and select 'Format' from the context menu. See *Format Dialog Box* for more details.

When a Crosstab is used to represent hierarchical cube dimensions, the column headers may display all of the hierarchical components within brackets. To override this behavior, manually format the column headers as described. Alternatively, ask an administrator to set the property `olap.table.originalContent=false` in the *sree.properties* file.

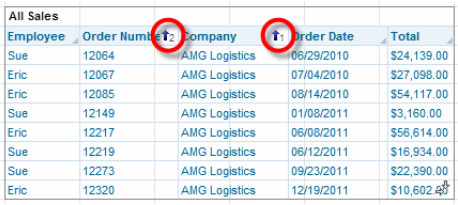
3.1.13 Sorting Table Columns

To sort a table column, hover the mouse over the right side of the column header. This enables the 'Sort Column' button. Click the button once to sort the column's data in ascending (alphabetical) order, click a second time to sort in descending (reverse alphabetical) order, and click a third time to restore the original order.



To sort multiple columns simultaneously (nested sorting), hold down the Ctrl-key as you click the 'Sort Column' button for the additional columns to be sorted. The order in which you click the 'Sort Column' button for the columns determines the nesting of the sorting.

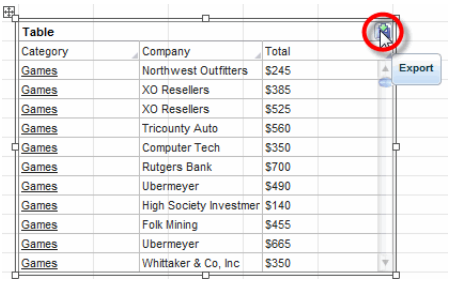
For example, click 'Sort Column' for a table's 'Company' column and then Ctrl-click 'Sort Column' for the 'Order Number' column to sort the 'Order Number' data on a company-by-company basis. The column header displays a small number to indicate the sorting precedence.



3.1.14 Exporting Table Data for Viewing

To export the data contained in a Table or Crosstab to an Excel file, follow the steps below:

1. Click the 'Export' button in the Table title bar. This opens the 'Export' dialog box.



2. In the 'Export' dialog box, enter a name for the exported Excel file, and press 'OK'.

This will save the table data as an Excel file in the browser's default download location.

3.1.15 Exporting Table Data for Editing

Viewsheets provide several features to assist in *offline data editing*. A user can export data from a Viewsheet to an Excel file, edit the data values within the Excel file, and re-import the revised data into the Viewsheet at a later time. This gives a user the ability to edit a Viewsheet "offline," while not actively logged into the server.

```
Note: Offline editing features are available only if product is
licensed for Viewsheet Forms.
```

To configure a Viewsheet so that users can export data to an Excel file for editing (and subsequent importing), follow the steps below:

1. For each table that you wish to make editable in Excel, right-click the table, and select 'Properties' from the context menu. This opens the 'Table Properties' dialog box.

2. Select the **General** tab, and enter a unique name for the table in the 'Name' field. (This name will be used to label the table in the Excel export, and to match the Excel table to the Viewsheet table during subsequent importing.)

3. Select the **Advanced** tab, and check the 'Enable Table Editing' option. Select the desired editing features, 'Add Rows', 'Delete

Rows', and/or 'Edit Rows'. See *Table Properties: Advanced Tab* for more information about these options.

4. Press 'OK' to close the 'Table Properties' dialog box.

See Also

Table Properties: Advanced Tab, for information on how to make a table editable.

Importing Excel Data into a Dashboard, for information on how to import the generated Excel file.

Exporting an Editable Dashboard, for information on how to export the Viewsheet.

3.2 Freehand Tables

The basic methods for presenting tabular data in a Viewsheet are *Tables* and *Crosstabs*. However, in some cases Tables and Crosstabs may not be flexible enough to create the specific tabular layout that you require.

For such cases, you can gain greater flexibility by using a *Freehand Table*. A Freehand Table allows you to specify complex cell expansions and associations, and integrate data from multiple Data Blocks into a single table.

3.2.1 Creating a Freehand Table

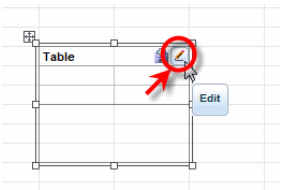
There are two ways to create Freehand Tables, either as a new component or by converting an exiting Table or Crosstab. The following sections explain the two methods:

Creating a Freehand Table as a New Component

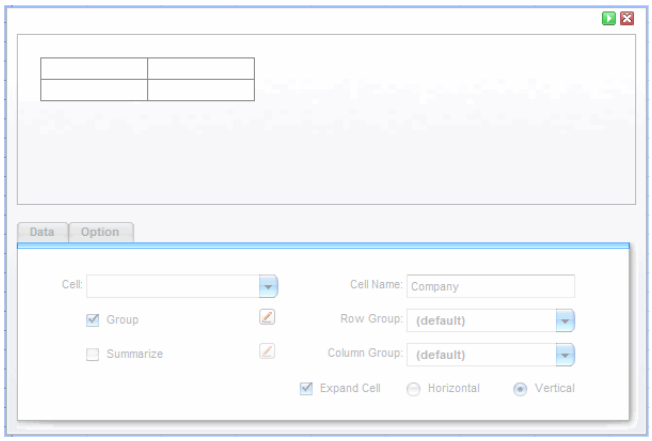
To create a new Freehand Table as a new component, follow the steps below:

1. Drag a Freehand Table from the Component panel into the Viewsheet. This adds a blank Freehand Table to the Viewsheet.

2. Press the 'Edit' button in the top-right corner of the Freehand Table.

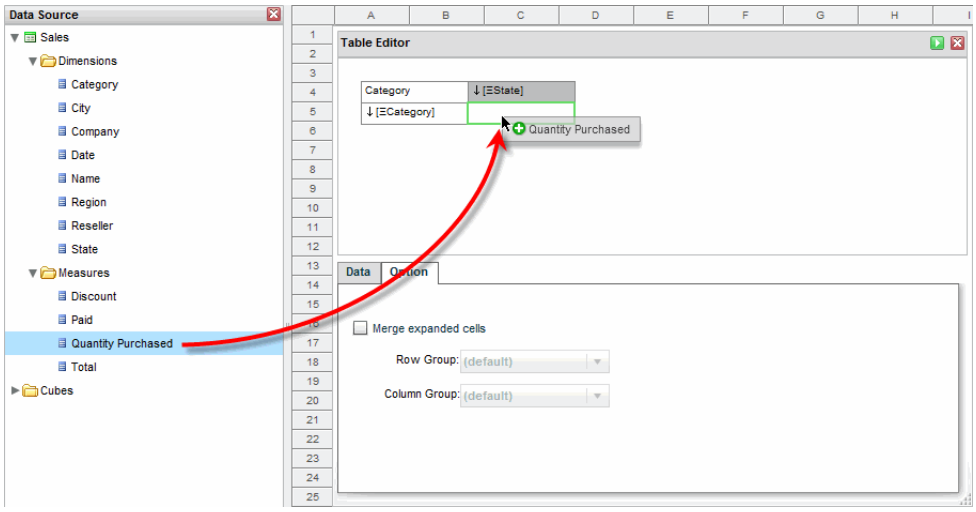


This opens the binding dialog box for the Freehand Table. The dialog box displays an interactive table diagram in the top panel, as well as a **Data** panel and **Options** panel below.

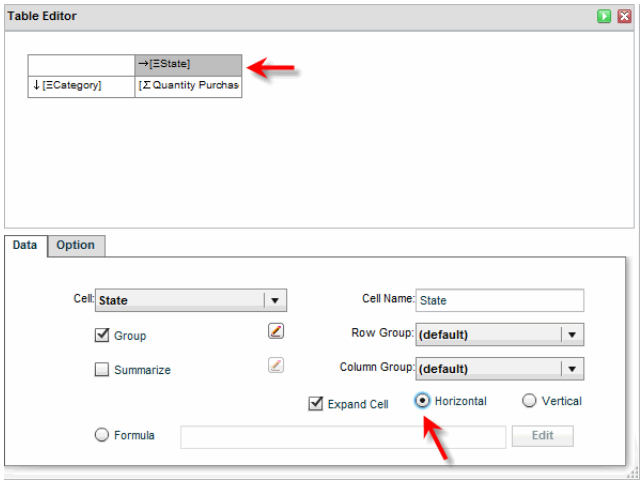


A field name is not permitted to contain the "?" symbol when used in a Freehand Table. To display data from such a field, first create a copy of the field as a Calculated Field (see *Calculated Fields*) or as an Expression Column (see *Creating an Expression/Formula Column* in *Advanced Data Mashup*) with a legal name.

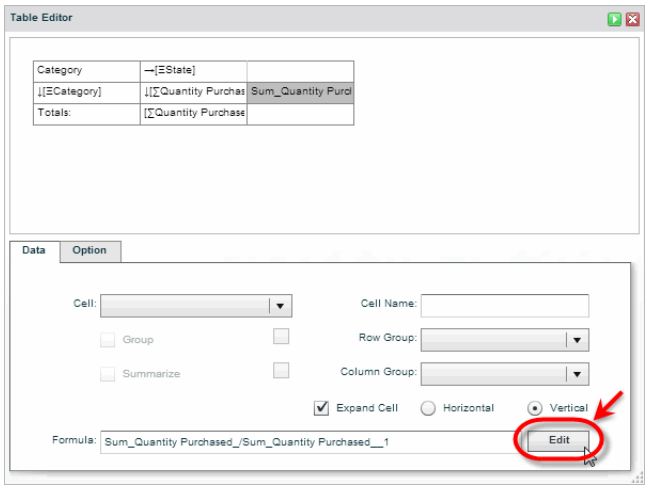
3. Drag the desired fields from the available Data Blocks into the table diagram in the binding dialog box.



4. Adjust the expansion directions and other properties for the individual cells as desired.

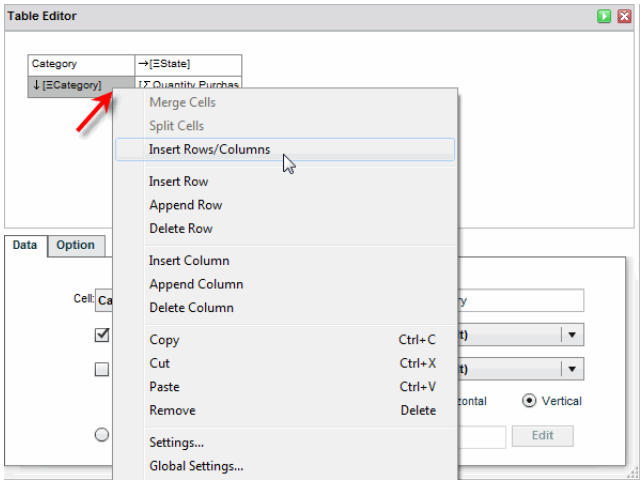


5. To populate a cell using script, press the 'Edit' button at the bottom of the dialog box. This opens the Script Editor and allows you to create a calculation based on cell values, parameter values, or component values.



Refer to the value in a cell by using syntax "`$cell_name`", where `cell_name` is the name assigned in the 'Cell Name' field for the cell that you are referencing.

6. To add rows and columns, right-click on a cell in the table diagram and select the desired option from the context menu.

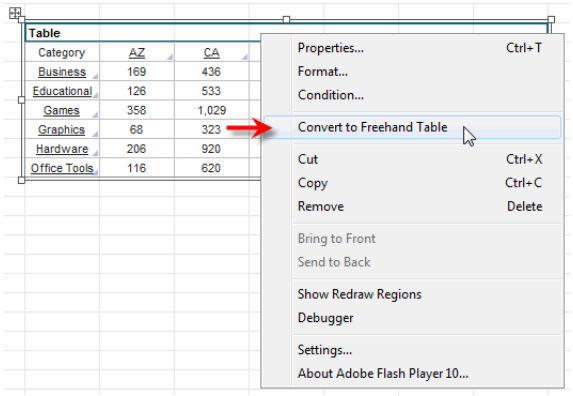


For more detailed information on how to configure expansion, grouping, and other properties, see *Walkthrough: Freehand Table*.

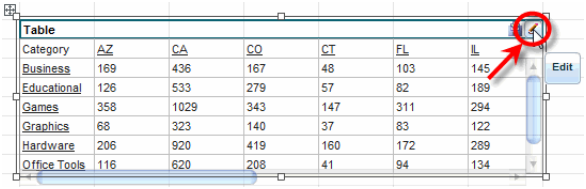
Creating a Freehand Table from an Existing Table or Crosstab

To create a a Freehand Table from an existing Table or Crosstab, follow the steps below:

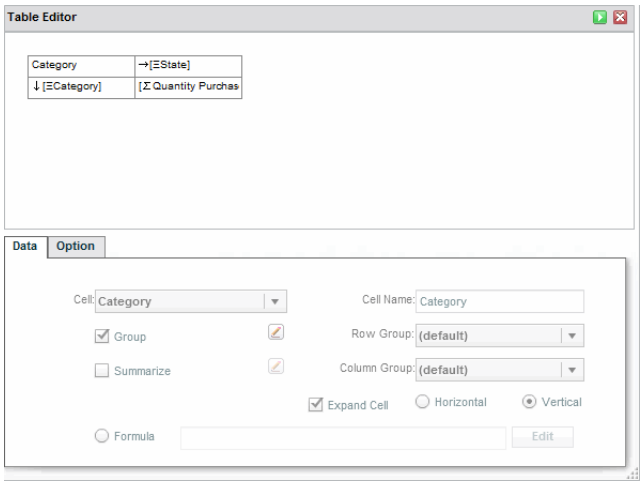
1. Right-click on the existing Table or Crosstab, and select ‘Convert to Freehand Table’.



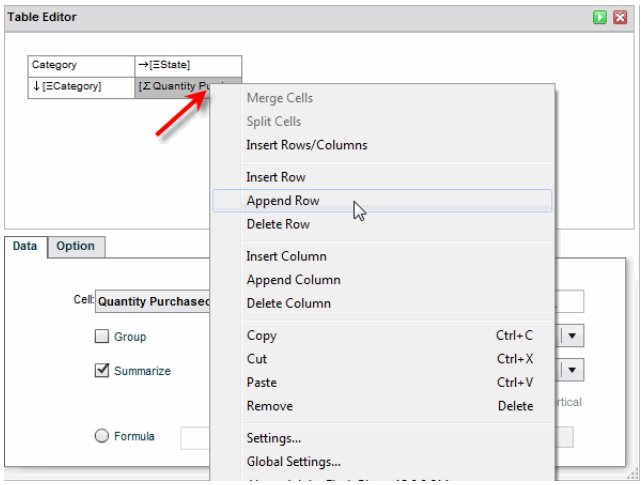
2. Click the ‘Edit’ button in the top-right corner of the new Freehand Table.



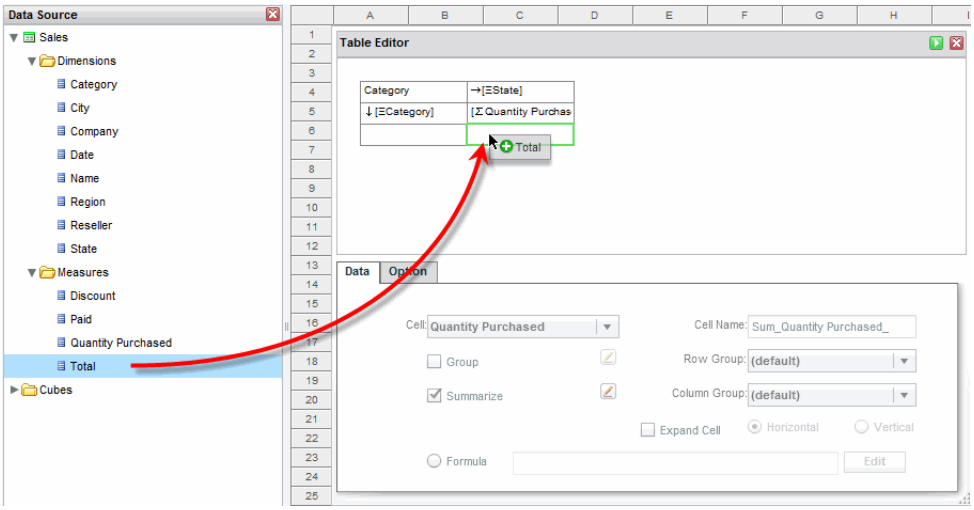
This opens the binding dialog box for the Freehand Table. The dialog box displays an interactive table diagram in the top panel, as well as a **Data** panel and **Options** panel below.



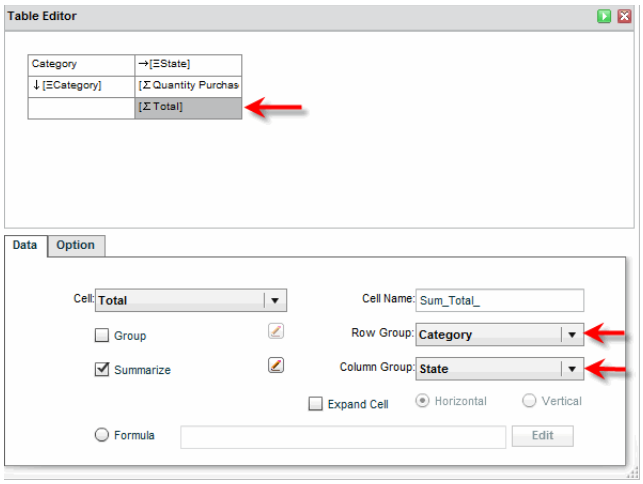
3. Add rows and columns as desired. To do this, right-click on a cell in the table diagram and select the desired option from the context menu.



4. Drag any additional desired fields from the available Data Blocks into the table diagram in the binding dialog box.



5. Adjust the expansion directions and other properties for the individual cells as desired.



For more detailed information on how to configure expansion, grouping, and other properties, see *Walkthrough: Freehand Table*.

See Also

Creating a Table, for information on creating a regular table.
Creating a Crosstab, for information on creating a regular crosstab.

3.2.2 Walkthrough: Freehand Table

The following walkthrough illustrates how you can create a complex data layout by using a freehand table.

Walkthrough

In this example, you will create a custom crosstab that breaks down sales totals both according to individual ‘State’ and ‘Category’ combinations,

and simultaneously according to 'Order Date' and 'Category' combinations.

In a regular Crosstab, the data is represented with a nested header structure, such as shown below. Note that the aggregated measure ('Total') is represented for every combination of 'State', 'Date', and 'Category'.

	AZ				CA				CO			
	2008	2009	2010	2011	2008	2009	2010	2011	2008	2009	2010	2011
Business	$85,625	$19,500	$53,250	$113,750	$167,750	$183,250	$247,375	$140,750	$112,500	$120,500	$3,750	$81,375
Educational	$5,942	$3,231	$3,995	$3,530	$25,919	$10,392	$16,293	$13,627	$11,462	$5,716	$9,565	$14,838
Games	$7,033	$13,463	$5,701	$13,266	$24,975	$13,111	$29,795	$24,879	$6,261	$5,618	$8,570	$11,603
Graphics	$14,376	$49		$10,450	$12,742	$13,325	$22,713	$25,664	$4,400	$28,333	$14,730	$980
Hardware	$68,840	$6,100	$760	$7,170	$28,480	$205,620	$107,190	$89,510	$95,940	$87,220	$32,350	$104,040
Office Tools	$7,490	$4,706		$4,032	$23,809	$26,236	$37,962	$51,342	$4,180	$14,274	$6,975	$11,887
Personal	$1,450	$9,088		$5,301	$35,340	$22,732	$45,081	$40,786	$10,662	$9,181	$10,524	$16,364

Instead, in this example you will create a table where the 'Date' headers are not nested within the 'State' headers, but are rather displayed separately, as shown below.

Category	AZ	CA	CO	CT	FL	IL	MA	MD	NJ	NV	2008	2009	2010	2011
Business	272125	739125	318125	49750	242625	301750	957625	690875	1462625	202625	1961500	2071250	1894375	2166375
Educational	16698	66231	41581	4883	7734	17119	71215	50187	126055	11106	134998	183837	131248	175417
Games	39463	92760	32052	12335	29921	26482	96547	59051	98186	13521	172330	185884	189400	201599
Graphics	24875	74444	48443	19849	25109	46559	110378	62644	178675	22566	195529	316686	199193	200287
Hardware	82870	430800	319550	36110	20130	82320	610960	314450	1055030	102000	995040	1289310	1456350	1231610
Office Tools	16228	139349	37316	5245	24882	26767	120292	133333	212760	35965	220541	320372	251236	334689
Personal	15839	143939	46731	34145	40240	58504	168501	95168	180642	9837	247885	281887	330693	317333

To create this table, begin by creating a regular Crosstab that contains just the 'State' and 'Category' combinations. You will then convert this table to a Freehand Table, and use basic Freehand Table techniques to add the 'Date' headers.

Follow the steps below:

1. Press the 'New Viewsheet' button in the Visual Composer toolbar. This opens the 'New Viewsheet' dialog box.

2. Expand the 'Data Source' node, and then expand the 'Orders' sub-node. Select the 'Order Model' data model, and press 'OK'. This creates a new Viewsheet based on the 'Order Model' data model.

3. Drag a 'Crosstab' component from the **Components** panel into the Viewsheet.

4. Press the 'Edit' button at the top-right of the Crosstab to open the binding dialog box.

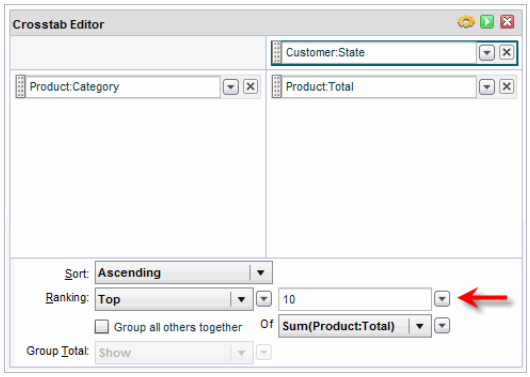
5. From the **Data Source** panel, drag fields from the 'Order Model' into the binding dialog box in the following way:

    a. Drag the 'Product.Category' field to the 'Row Header' panel.

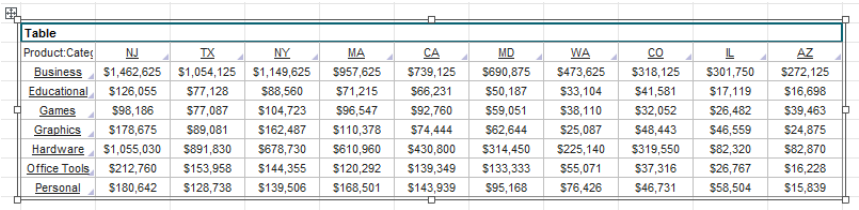
    b. Drag the 'Customer.State' field to the 'Column Header' panel.

c. Drag the ‘Product.Total’ field to the ‘Measure’ panel.

d. Click the Customer.State’ field in the ‘Column Header’ panel to select it. From the ‘Ranking’ menu in the bottom panel, select ‘Top’. Enter ‘10’ as the number of groups. (This will return the top 10 groups according to the summed total.)



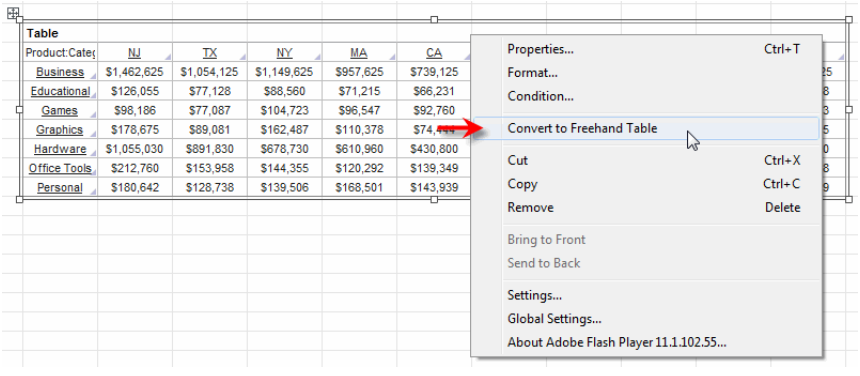
e. Press ‘Apply’ to close the binding dialog box. This creates a new Crosstab in the Viewsheet.



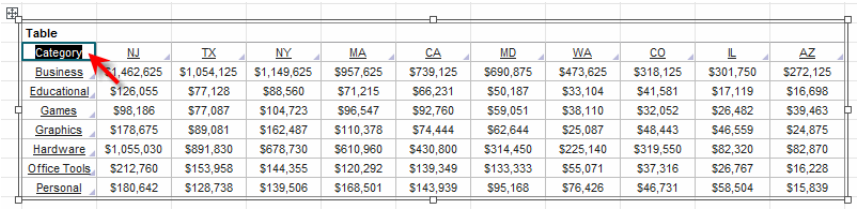
Table

Product:Cate	NJ	TX	NY	MA	CA	MD	WA	CO	IL	AZ
Business	$1,462,625	$1,054,125	$1,149,625	$957,625	$739,125	$690,875	$473,625	$318,125	$301,750	$272,125
Educational	$126,055	$77,128	$88,560	$71,215	$66,231	$50,187	$33,104	$41,581	$17,119	$16,698
Games	$98,186	$77,087	$104,723	$96,547	$92,760	$59,051	$38,110	$32,052	$26,482	$39,463
Graphics	$178,675	$89,081	$162,487	$110,378	$74,444	$62,644	$25,087	$48,443	$46,559	$24,875
Hardware	$1,055,030	$891,830	$678,730	$610,960	$430,800	$314,450	$225,140	$319,550	$82,320	$82,870
Office Tools	$212,760	$153,958	$144,355	$120,292	$139,349	$133,333	$55,071	$37,316	$26,767	$16,228
Personal	$180,642	$128,738	$139,506	$168,501	$143,939	$95,168	$76,426	$46,731	$58,504	$15,839

6. Right-click the Crosstab, and select ‘Convert to Freehand Table’ from the context menu.



7. Click the top-left corner cell of the table to select it. Type the text “Category” into the cell.

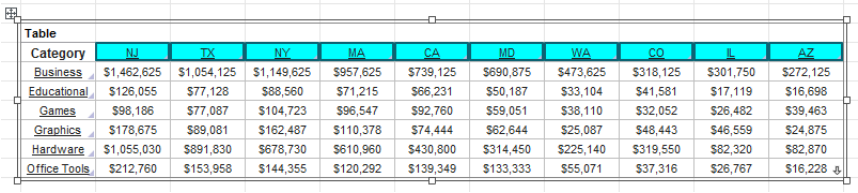


Table

Category	NJ	TX	NY	MA	CA	MD	WA	CO	IL	AZ
Business	$1,462,625	$1,054,125	$1,149,625	$957,625	$739,125	$690,875	$473,625	$318,125	$301,750	$272,125
Educational	$126,055	$77,128	$88,560	$71,215	$66,231	$50,187	$33,104	$41,581	$17,119	$16,698
Games	$98,186	$77,087	$104,723	$96,547	$92,760	$59,051	$38,110	$32,052	$26,482	$39,463
Graphics	$178,675	$89,081	$162,487	$110,378	$74,444	$62,644	$25,087	$48,443	$46,559	$24,875
Hardware	$1,055,030	$891,830	$678,730	$610,960	$430,800	$314,450	$225,140	$319,550	$82,320	$82,870
Office Tools	$212,760	$153,958	$144,355	$120,292	$139,349	$133,333	$55,071	$37,316	$26,767	$16,228
Personal	$180,642	$128,738	$139,506	$168,501	$143,939	$95,168	$76,426	$46,731	$58,504	$15,839

(Optional) Use the Visual Composer toolbar buttons to make the text large and bold.

8. Shift-click to select all of the ‘State’ cells in the header row. Press the ‘Fill Color’ toolbar button, and specify a light blue background.



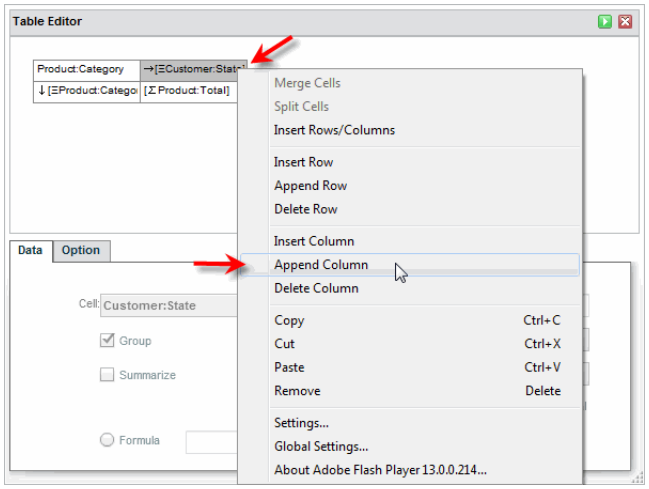
Table

Category	NJ	TX	NY	MA	CA	MD	WA	CO	IL	AZ
Business	$1,462,625	$1,054,125	$1,149,625	$957,625	$739,125	$690,875	$473,625	$318,125	$301,750	$272,125
Educational	$126,055	$77,128	$88,560	$71,215	$66,231	$50,187	$33,104	$41,581	$17,119	$16,698
Games	$98,186	$77,087	$104,723	$96,547	$92,760	$59,051	$38,110	$32,052	$26,482	$39,463
Graphics	$178,675	$89,081	$162,487	$110,378	$74,444	$62,644	$25,087	$48,443	$46,559	$24,875
Hardware	$1,055,030	$891,830	$678,730	$610,960	$430,800	$314,450	$225,140	$319,550	$82,320	$82,870
Office Tools	$212,760	$153,958	$144,355	$120,292	$139,349	$133,333	$55,071	$37,316	$26,767	$16,228

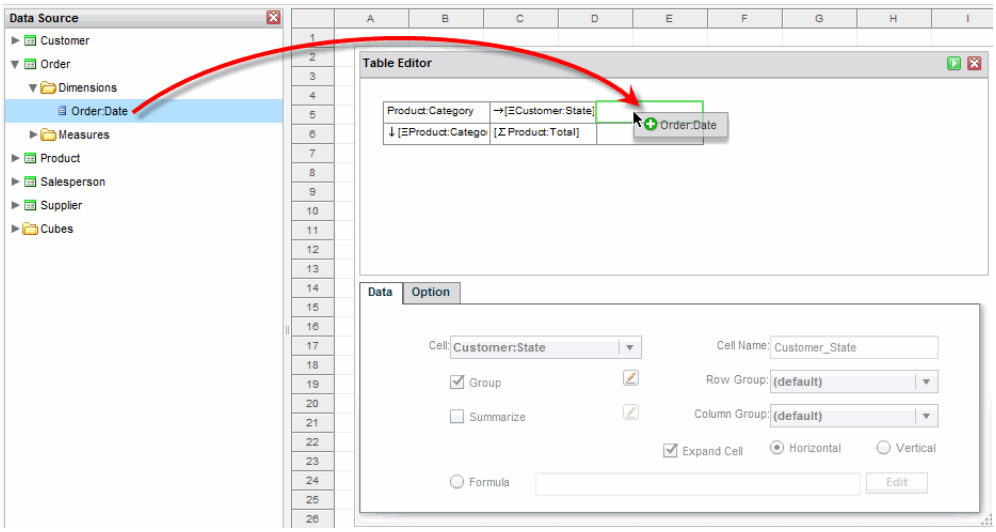
9. You will now use freehand table operations to add the ‘Date’ headers to the table. Follow the steps below:

   a. Press the ‘Edit’ button in the top-right corner of the Freehand Table. This opens the binding dialog box.

   b. Right-click the top-right corner cell and select ‘Append Column’ from the context menu. This adds a new column.

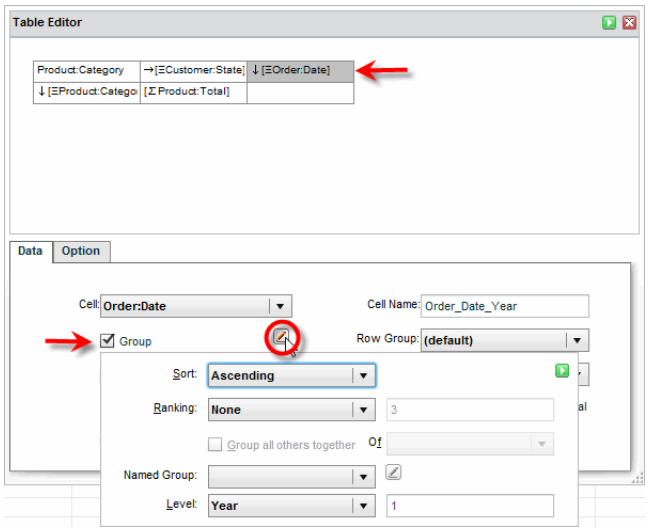


   c. From the ‘Order’ node in the **Data Source** panel, drag the ‘Date’ attribute to the new top-right corner cell of the table diagram. This adds the ‘Order.Date’ attribute as a column header.



d. Click to select this new ‘Order.Date’ cell. Next to the ‘Expand Cell’ option in the bottom panel, select ‘Horizontal’. This sets the year groups in the ‘Order.Date’ field to display horizontally, as multiple columns headers.

e. Select the ‘Group’ option and press the adjacent ‘Edit’ button. From the ‘Level’ menu, select ‘Year’ and press the green ‘Apply’ button.



f. Press the green ‘Apply’ button at the top of the binding dialog box to close the dialog box.

g. Expand the table so that you can see all columns. Shift-click to select all of the ‘Date’ headers. Press the ‘Fill Color’ button in the Visual Composer toolbar and set a yellow background for the ‘Date’ headers.

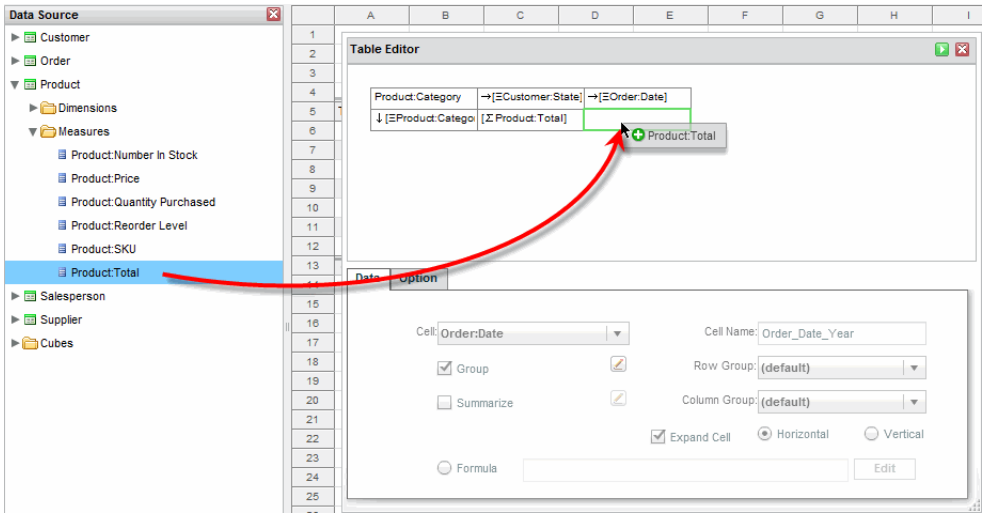
Table

Category	NJ	TX	NY	MA	CA	MD	WA	CO	IL	AZ	12/31/1969	12/31/1969	12/
Business	$1,462,625	$1,054,125	$1,149,625	$957,625	$739,125	$690,875	$473,625	$318,125	$301,750	$272,125			
Educational	$126,055	$77,128	$88,560	$71,215	$66,231	$50,187	$33,104	$41,581	$17,119	$16,698			
Games	$98,186	$77,087	$104,723	$96,547	$92,760	$59,051	$38,110	$32,052	$26,482	$39,463			
Graphics	$178,675	$89,081	$162,487	$110,378	$74,444	$62,644	$25,087	$48,443	$46,559	$24,875			
Hardware	$1,055,030	$891,830	$678,730	$610,960	$430,800	$314,450	$225,140	$319,550	$82,320	$82,870			
Office Tools	$212,760	$153,958	$144,355	$120,292	$139,349	$133,333	$55,071	$37,316	$26,767	$16,228			

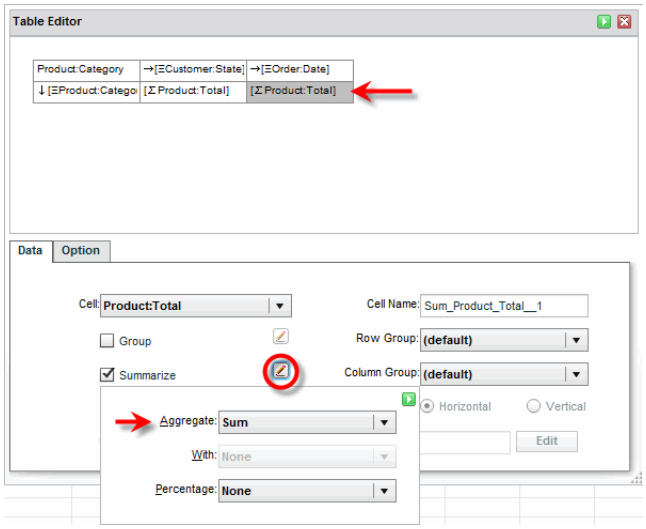
10. You will now use freehand table operations to add an aggregate 'Total' measure corresponding to the year groups. Follow the steps below:

    a. Press the 'Edit' button in the top-right corner of the Freehand Table to reopen the binding dialog box.

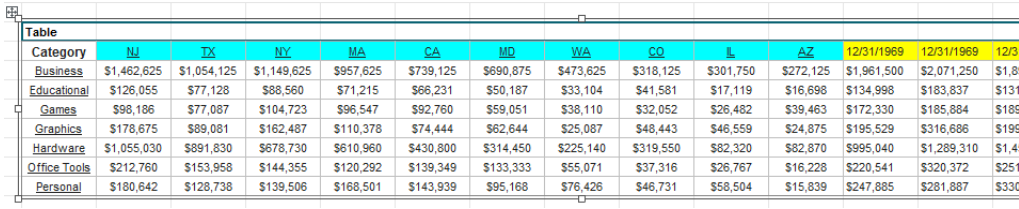
    b. In the **Data Source** panel, expand the 'Product' node, and drag the 'Total' attribute to the bottom-right corner cell of the table diagram. This adds the 'Product.Total' attribute as a measure.



    c. Select this new 'Product.Total' cell, and enable the 'Summarize' option in the panel below. Press the adjacent 'Edit' button, select 'Sum' from the 'Aggregate' menu, and press the green 'Apply' button. This will aggregate the totals for each combination of 'Category' and 'Date'.



d. Press the green 'Apply' button to close the binding dialog box.



Table

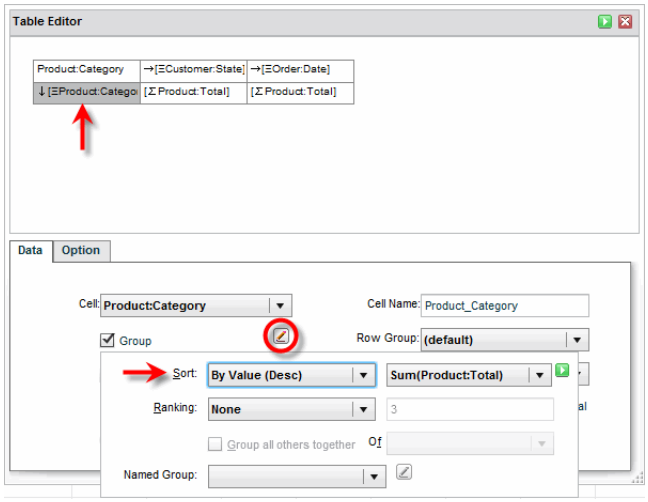
Category	NJ	TX	NY	MA	CA	MD	WA	CO	IL	AZ	12/31/1969	12/31/1969	12/3
Business	$1,462,625	$1,054,125	$1,149,625	$957,625	$739,125	$690,875	$473,625	$318,125	$301,750	$272,125	$1,961,500	$2,071,250	$1,8
Educational	$126,055	$77,128	$88,560	$71,215	$66,231	$50,187	$33,104	$41,581	$17,119	$16,698	$134,998	$183,837	$131
Games	$98,186	$77,087	$104,723	$96,547	$92,760	$59,051	$38,110	$32,052	$26,482	$39,463	$172,330	$185,884	$189
Graphics	$178,675	$89,081	$162,487	$110,378	$74,444	$62,644	$25,087	$48,443	$46,559	$24,875	$195,529	$316,686	$199
Hardware	$1,055,030	$891,830	$678,730	$610,960	$430,800	$314,450	$225,140	$319,550	$82,320	$82,870	$995,040	$1,289,310	$1,4
Office Tools	$212,760	$153,958	$144,355	$120,292	$139,349	$133,333	$55,071	$37,316	$26,767	$16,228	$220,541	$320,372	$251
Personal	$180,642	$128,738	$139,506	$168,501	$143,939	$95,168	$76,426	$46,731	$58,504	$15,839	$247,885	$281,887	$330

11. Observe that the 'Category' row headers are currently sorted alphabetically (e.g., Business, Educational, Games, etc.). To sort the categories by the value of the corresponding aggregate total, follow the steps below:

    a. Press the 'Edit' button in the top-right corner of the Freehand Table to reopen the binding dialog box.

    b. Click to select the grouped 'Category' cell (bottom-left cell in the table diagram).

    c. Press the 'Edit' button adjacent to the 'Group' option. From the 'Sort' menu, select the option 'By Value (Desc)', and press the green 'Apply' button.



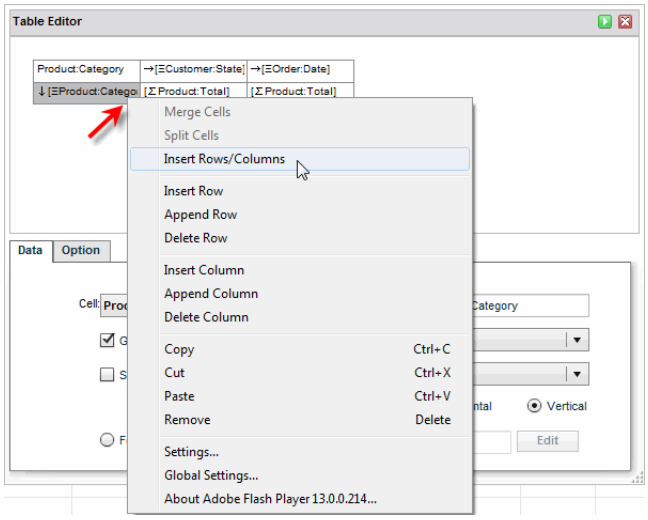
d. Press the green 'Apply' button in the binding dialog box to close the dialog box. Observe that the categories are now shown in descending order according to the totals (e.g., Business, Hardware, Personal, etc.).

Table													
Category	NJ	TX	NY	MA	CA	MD	WA	CO	IL	AZ	12/31/1969	12/31/1969	12/31/1969
Business	$1,462,625	$1,054,125	$1,149,625	$957,625	$739,125	$690,875	$473,625	$318,125	$301,750	$272,125	$1,961,500	$2,071,250	$1,894,375
Hardware	$1,055,030	$891,830	$678,730	$610,960	$430,800	$314,450	$225,140	$319,550	$82,320	$82,870	$995,040	$1,289,310	$1,456,350
Personal	$180,642	$128,738	$139,506	$168,501	$143,939	$95,168	$76,426	$46,731	$58,504	$15,839	$247,885	$281,887	$330,693
Office Tools	$212,760	$153,958	$144,355	$120,292	$139,349	$133,333	$55,071	$37,316	$26,767	$16,228	$220,541	$320,372	$251,236
Graphics	$178,675	$89,081	$162,487	$110,378	$74,444	$62,644	$25,087	$48,443	$46,559	$24,875	$195,529	$316,686	$199,193
Games	$98,186	$77,087	$104,723	$96,547	$92,760	$59,051	$38,110	$32,052	$26,482	$39,463	$172,330	$185,884	$189,400
Educational	$126,055	$77,128	$88,560	$71,215	$66,231	$50,187	$33,104	$41,581	$17,119	$16,698	$134,998	$183,837	$131,248

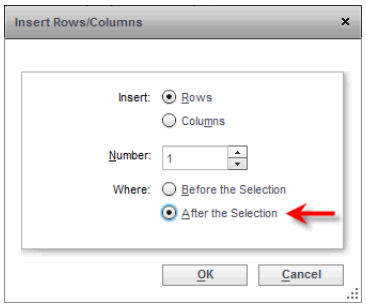
12. Add a row of grand totals at the bottom of the Freehand Table. Follow the steps below:

a. Press the 'Edit' button in the top-right corner of the Freehand Table to reopen the binding dialog box.

b. Right-click on any cell in the bottom row of the table diagram, and select 'Insert Rows/Columns'. This opens the 'Insert Rows/ Columns' dialog box.

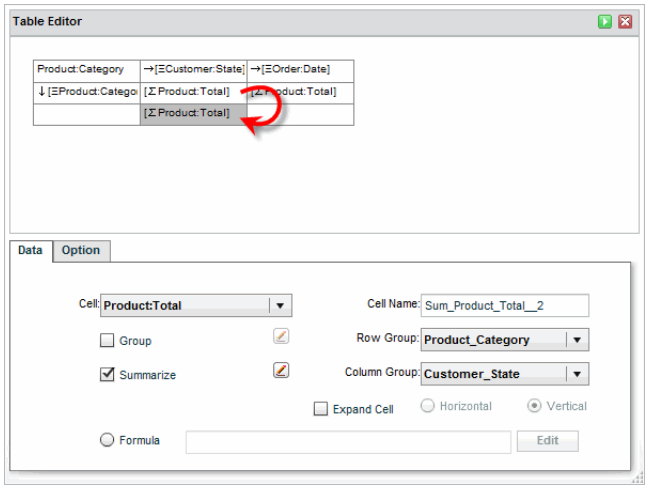


c. Configure the settings to insert one row ‘After the Selection’, and press ‘OK’. This inserts a new row into the table diagram.



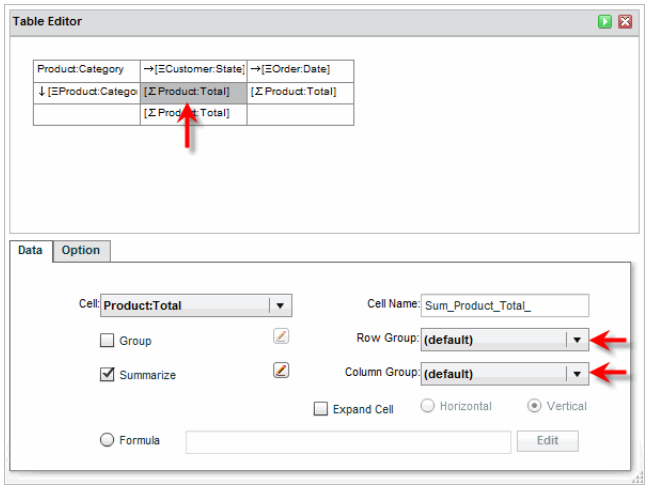
d. Right-click on one of the ‘Product:Total’ cells, and select ‘Copy’ from the context menu.

e. Right-click in the empty cell below one of the ‘Product:Total’ cells, and select ‘Paste’ from the context menu. This places a copy of the ‘Product:Total’ field into that cell.



f. Repeat the above step for the empty below the other ‘Product:Total’ cell.

g. Click to select the ‘Product.Total’ cell at the center of the table diagram. In the ‘Row Group’ and ‘Column Group’ menus, select the ‘default’ options, and press the green ‘Apply’ button.



The ‘Row Group’ and ‘Column Group’ settings for a cell determine how the cell will respond to the expansion of other cells. To anchor the selected cell to a expanding grouping cell (so that the selected cell will observe the same grouping as it expands), select the name of the desired grouping cell in the ‘Row Group’ or ‘Column Group’ menu.

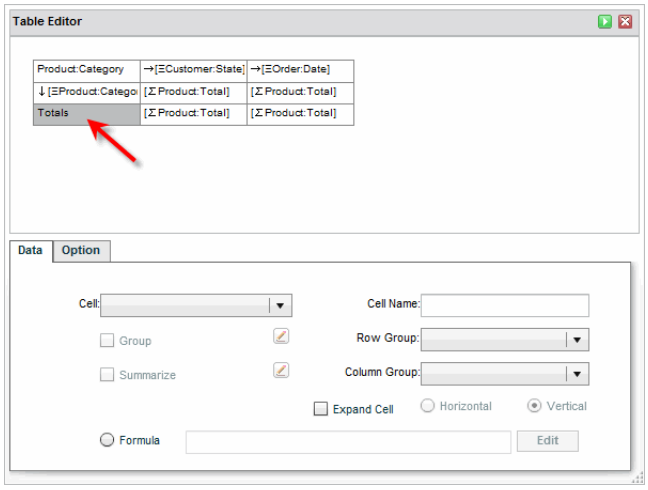
The ‘default’ option for ‘Row Group’ selects the nearest candidate grouping cell (if one exists) in the same row and to the left of the selected cell. In this example, the ‘Product.Category’ cell provides a candidate row grouping cell in the same row as

the selected ‘Product.Total’ field. Therefore, in this case the ‘default’ option in the ‘Row Group’ menu is equivalent to ‘Product.Category’.

The ‘default’ option for ‘Column Group’ selects the nearest candidate grouping cell (if one exists) in the same column and above the selected cell. In this example, the ‘Customer.State’ cell provides a candidate column grouping cell in the same column as the selected ‘Product.Total’ field. Therefore, in this case the ‘default’ option in the ‘Column Group’ menu is equivalent to ‘Customer.State’.

h. Specify the same ‘default’ options for the other ‘Product.Total’ cells.

i. Double-click in the bottom-left cell of the table, and enter the text “Totals”.



13. Expand the table as needed to show all rows and columns.

Table													
Category	NJ	TX	NY	MA	CA	MD	WA	CO	IL	AZ	12/31/1969	12/31/1969	12/31/1969
Business	$1,462,625	$1,054,125	$1,149,625	$957,625	$739,125	$690,875	$473,625	$318,125	$301,750	$272,125	$1,961,500	$2,071,250	$1,894,375
Hardware	$1,055,030	$891,830	$678,730	$610,960	$430,800	$314,450	$225,140	$319,550	$82,320	$82,870	$995,040	$1,289,310	$1,456,350
Personal	$180,642	$128,738	$139,506	$168,501	$143,939	$95,168	$76,426	$46,731	$58,504	$15,839	$247,885	$281,887	$330,693
Office Tools	$212,760	$153,958	$144,355	$120,292	$139,349	$133,333	$55,071	$37,316	$26,767	$16,228	$220,541	$320,372	$251,236
Graphics	$178,675	$89,081	$162,487	$110,378	$74,444	$62,644	$25,087	$48,443	$46,559	$24,875	$195,529	$316,686	$199,193
Games	$98,186	$77,087	$104,723	$96,547	$92,760	$59,051	$38,110	$32,052	$26,482	$39,463	$172,330	$185,884	$189,400
Educational	$126,055	$77,128	$88,560	$71,215	$66,231	$50,187	$33,104	$41,581	$17,119	$16,698	$134,998	$183,837	$131,248
Totals	$3,313,973	$2,471,947	$2,467,986	$2,135,518	$1,686,648	$1,405,708	$926,563	$843,798	$559,501	$468,098	$3,927,823	$4,649,226	$4,452,495

3.2.3 Freehand Table Properties

The Freehand Table component’s ‘Properties’ dialog box provides **General**, **Advanced**, and **Script** tabs. The next sections discuss the component-specific properties available under these tabs.

See Also

Table Properties, for information about regular Table properties.
Properties Dialog Box, for information about common component properties.
Format Dialog Box, for information on the 'Format' dialog box.

Freehand Table Properties: General Tab

The **General** tab in the 'Freehand Table Properties' dialog box is the same as for regular tables.

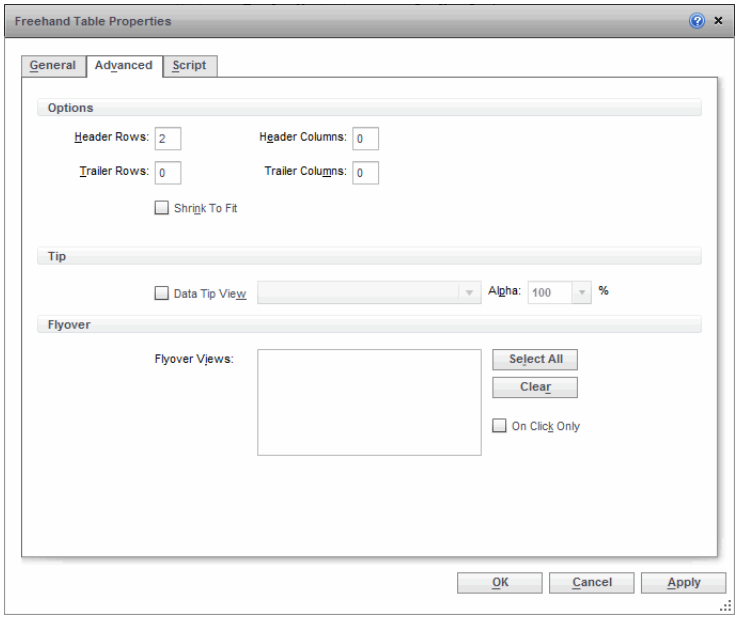
See Also

General Tab, for information about common properties in the **General** tab
Table Properties, for information about general table properties.

Freehand Table Properties: Advanced Tab

The **Advanced** tab in the 'Freehand Table Properties' dialog box provides the following freehand table-specific properties.

Header Rows	The number of initial rows that should be treated as header rows when a table style is applied.
Header Columns	The number of initial columns that should be treated as header columns when a table style is applied.
Trailer Rows	The number of final rows that should be treated as trailer rows when a table style is applied.
Trailer Columns	The number of final columns that should be treated as trailer columns when a table style is applied.
Shrink to Fit	When enabled, this reduces the height of the table to match the number of records displayed. For example, if the original table height (as designed) was 10 rows, and the user then makes selections which restrict the result set to five rows, the bottom table border retracts or "shrinks" to eliminate any empty rows at the bottom of the table.
Data Tip View	A Viewsheet component is displayed when mouse hovers over table area. The component is filtered based on the hover region. See *Data Tip View* for more details.
Flyover Views	Allows you to select a set of Viewsheet components which will be adaptively filtered as the user hovers the mouse or clicks on the crosstab area. (Select 'On Click Only' if you want the filtered components to update when the user clicks.) Filtering is based on the grouping fields (dimensions) in the row and column on which the user hovers or clicks. See *Flyover View* for more details.



3.3 Crosstabs

A crosstab, also known as a pivot table, contains one or more column headers and one or more row headers, and displays aggregated measure values at the row-column intersections.

For example, a train schedule displays dates horizontally across (column header), station names vertically down (row header), and departure times (measure) at the intersections. Not every row-column combination need have an associated measure value; for example, there may be combinations of date and station for which there is no corresponding train departure.

In general, the values at the row-column intersections of a crosstab represent *summary* information of a measure, rather than point values. For example, a crosstab with row headers representing 'Salesperson' and column headers representing 'Month of Year' might contain values at the intersections representing 'Average Sale Price' or 'Maximum Sale Price'. The measure in both cases is 'Sale Price'. The summarization method is, respectively, Average or Maximum. By effectively using summarization, crosstabs can display large amounts of data in a compact form.

3.3.1 Creating a Crosstab

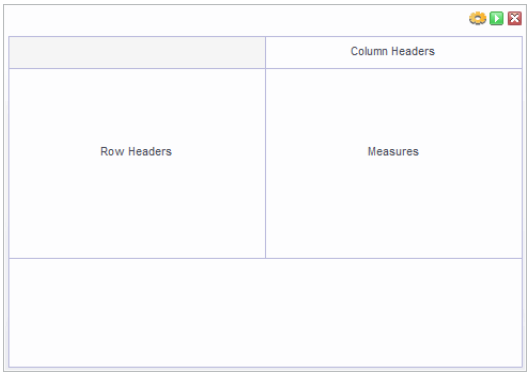
To add a crosstab to a Viewsheet, follow the steps below:

1. Drag the Crosstab element from the component tree into the Viewsheet grid.

2. Move the mouse over the top right corner of the crosstab, and click the 'Edit' button.



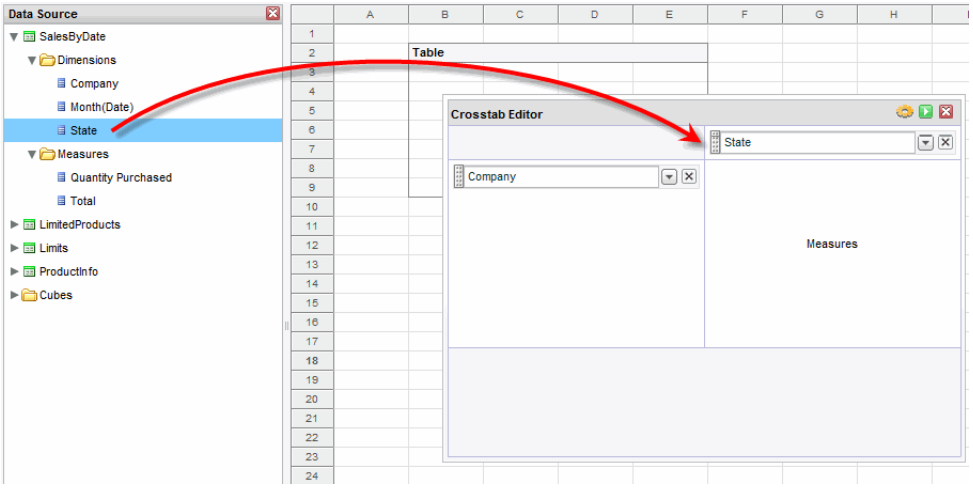
This opens the crosstab data dialog box.



The panel displays all OLAP cubes defined on the data source.

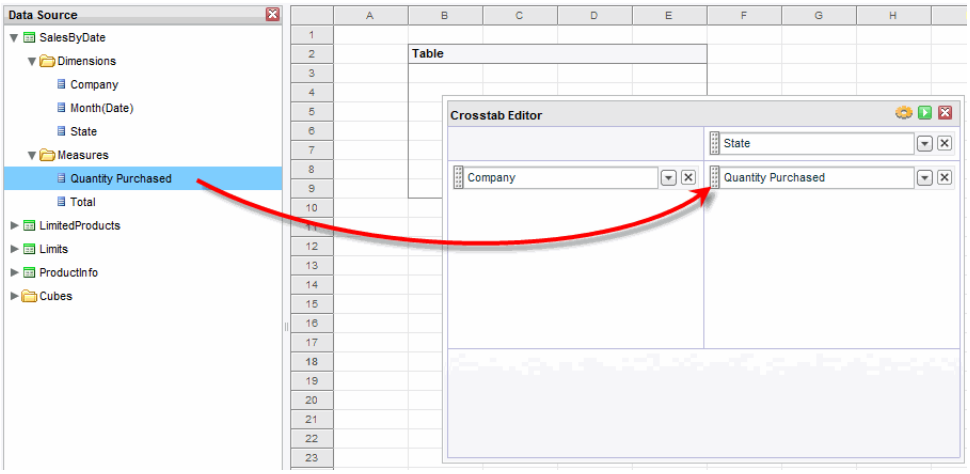
3. In the **Data Source** panel, expand a Data Block, query, data model, or OLAP cube (hierarchical model) to display the available dimensions and measures.

4. Drag the desired columns or cube dimensions into the ‘Column Headers’ and ‘Row Headers’ regions.



You can change the order of multiple headers or measures by dragging them up or down.

5. Drag the desired columns or cube measures into the region labeled ‘Measures’.



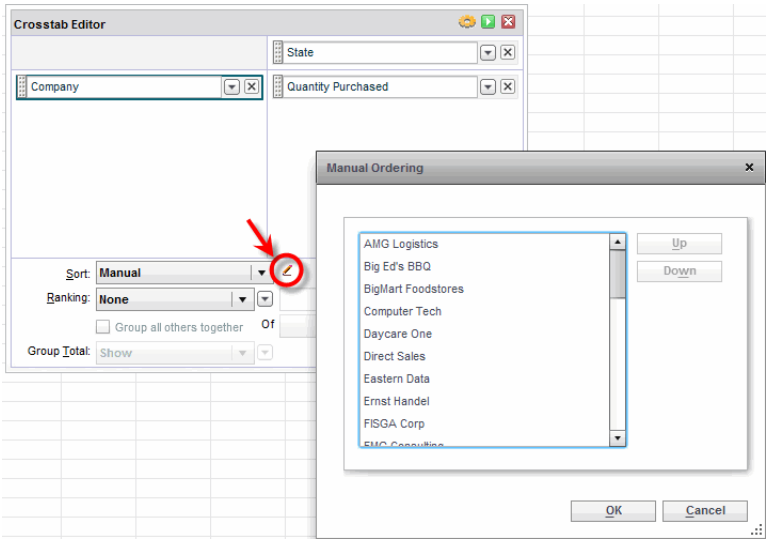
6. (Optional) Specify group totals and ranking:

   a. Click on a row header or column header. This activates the ‘Sort’, ‘Ranking’, and ‘Group Total’ controls.

   b. In the ‘Sort’ menu, specify how the values in the field should be ordered. ‘Ascending’ and ‘Descending’ sort the values in alphabetical or reverse-alphabetical order. ‘By Value (Asc)’ and ‘By Value (Desc)’ sort the values according to the aggregate measure selected in the adjacent menu.

      To specify an arbitrary sort order, select the ‘Manual’ option in the ‘Sort’ menu, and press the ‘Edit’ button to open the ‘Manual Ordering’ dialog box. To reorder an item in the ‘Manual Ordering’ dialog box, drag the item up or down within the list or select the item and press the ‘Up’ or ‘Down’ buttons. Press ‘OK’ to close the dialog box.

For end-user sorting features, see *Sorting a Table*.



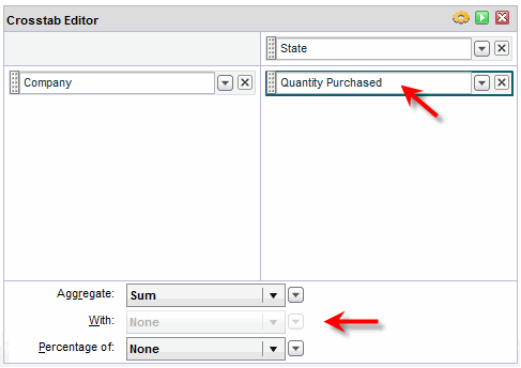
c. In the ‘Group Total’ menu, specify whether group totals should be shown.

d. In the ‘Ranking’ menu, Select the ‘Top’ or ‘Bottom’ filter, and specify the (integer) number of top or bottom groups to select. In the adjacent menu, specify the measure by which groups should be ranked.

e. Select ‘Group all others together’ to create an “Others” category which represents (in aggregate) all the groups excluded by the Top/Bottom-N filter.

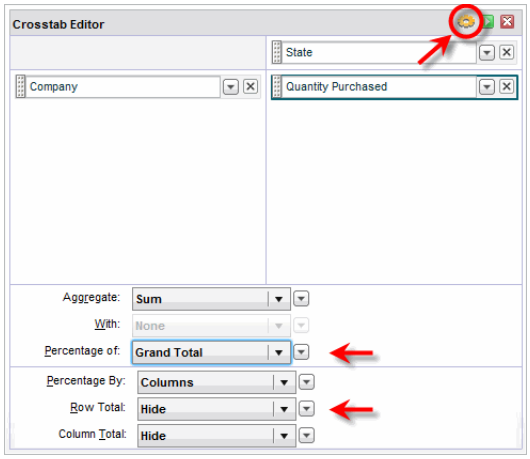
These controls are disabled for OLAP cubes. Cube aggregations are defined in the model itself. (See *OLAP Overlay – Multi-dimensional Analysis* in the *Advanced Data Modeling*.)

7. Click on a measure, and select the summarization method from the ‘Aggregate’ menu.



a. To display a univariate measure (‘Sum’, ‘Count’, etc.) as a percentage, select the basis for percentage in the ‘Percentage of’ menu.

b. If you choose a percentage representation, click the ‘Options’ button (at the top of the dialog box) to select the dimension along which the percentage is computed (‘Rows’ or ‘Columns’). Percentage by rows uses the row totals, and by columns uses the column totals.



c. To display a bivariate aggregation measure (e.g., 'Correlation', 'Weighted Average', etc.), select the second operand (column) from the 'with' menu.

8. Click on the 'Apply' button (the arrow on the top right) to close the binding dialog box.

9. If desired, expand the crosstab horizontally so that all of the column headers are visible.

See Also

Percentage Representation for Aggregation, to display data as a percentage.

Crosstab Aggregation Methods for information on summarization options.

OLAP Overlay – Multidimensional Analysis, in *Advanced Data Modeling*, for general information on hierarchical overlays.

3.3.2 Crosstab Properties

The Crosstab component's 'Properties' dialog box provides **General**, **Advanced**, **Hierarchy**, and **Script** tabs. The next sections discuss the component-specific properties available under these tabs.

See Also

Table Properties, for information about regular Table properties.

Properties Dialog Box, for information about common component properties.

Format Dialog Box, for information on the 'Format' dialog box.

Crosstab Properties: General Tab

The **General** tab in the 'Crosstab Properties' dialog box is the same as for regular tables.

See Also

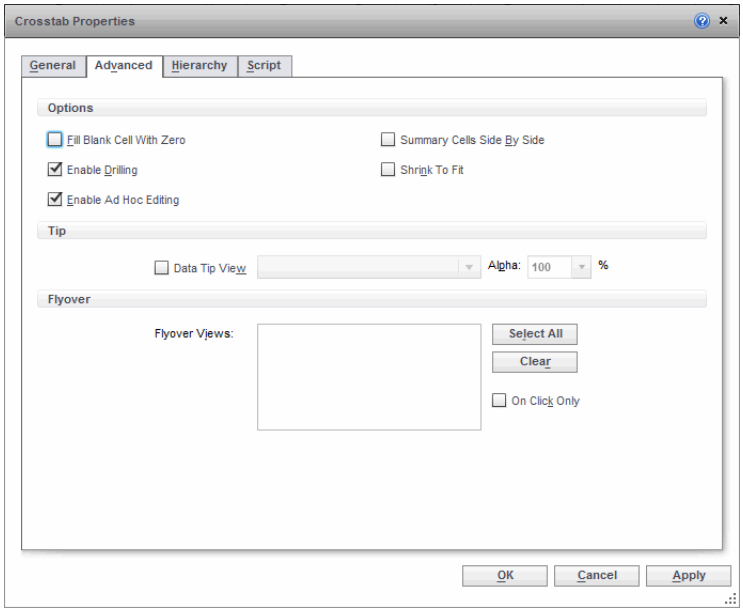
General Tab, for information about common properties in the **General** tab

Table Properties, for information about general table properties.

Crosstab Properties: Advanced Tab

The **Advanced** tab in the 'Crosstab Properties' dialog box provides the following crosstab-specific properties.

Fill Blank Cell With Zero	Places a zero value in measure cells corresponding to row and column heading combinations for which no records exist. Otherwise, these cells are simply left blank.
Summary Cells Side By Side	Places multiple aggregated measures side-by-side, rather than stacking them vertically.
Enable Drilling	Allows the user to drill up and down in date or dimension fields. See *Drilling Down into a Crosstab* for more information.
Enable Ad Hoc Editing	Allow end users to access the Crosstab Editor and edit the table.
Shrink to Fit	When enabled, this reduces the height of the table to match the number of records displayed. For example, if the original table height (as designed) was 10 rows, and the user then makes selections which restrict the result set to five rows, the bottom table border retracts or "shrinks" to eliminate any empty rows at the bottom of the table.
Data Tip View	A Viewsheet component is displayed when mouse hovers over crosstab area. The component is filtered based on the hover region. See *Data Tip View* for more details.
Flyover Views	Allows you to select a set of Viewsheet components which will be adaptively filtered as the user hovers the mouse or clicks on the crosstab area. (Select 'On Click Only' if you want the filtered components to update when the user clicks.) Filtering is based on the grouping fields (dimensions) in the row and column on which the user hovers or clicks. See *Flyover View* for more details.



Crosstab Properties: Hierarchy Tab

The **Hierarchy** tab in the 'Crosstab Properties' dialog box allows you to specify the hierarchy for a custom drill-down. The procedure for this is the same as that for a Chart. See *Chart Properties: Hierarchy Tab* for details.

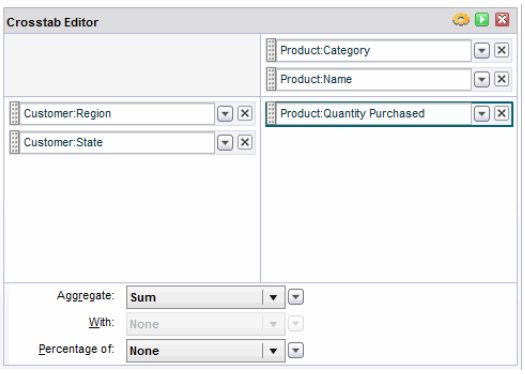
For information on how to navigate a crosstab with drill-downs, see *Drilling Down into a Crosstab*.

3.3.3 Crosstab Aggregation Methods

In general, there are many data records that correspond to a given row and column heading in the crosstab. The values of the measures for these records must therefore be aggregated to yield the single scalar value that appears at a given row-column intersection. The aggregation method specified by the 'Aggregate' menu determines the way in which this summarization is done.

Aggregation Example

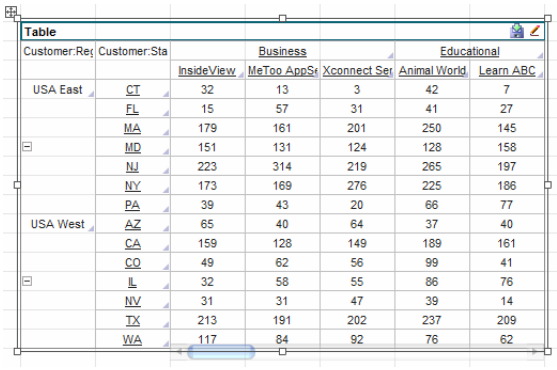
As an example of crosstab aggregation, consider the crosstab configuration below.



In this configuration:

- Row dimension is a ‘Region’/’State’ hierarchy
- Column dimension is a ‘Category’/’Product’ hierarchy
- Measure is ‘Quantity Purchased’

The ‘Aggregate’ menu specifies ‘Sum’, which indicates that each value in the crosstab will represent the *sum* of ‘Quantity Purchased’ taken over all records matching the given row and column headings. Thus, for the row region heading ‘USA East’ and the column category heading ‘Business’, the summary value of the measure shown in the crosstab represents the *summed quantity* of all ‘Business’ category products purchased by ‘USA East’ region customers.


Table

Customer:Re	Customer:Sta	Business			Educational	
		InsideView	MeToo AppSe	Xconnect Ser	Animal World	Learn ABC
USA East	CT	32	13	3	42	7
	FL	15	57	31	41	27
	MA	179	161	201	250	145
	MD	151	131	124	128	158
	NJ	223	314	219	265	197
	NY	173	169	276	225	186
	PA	39	43	20	66	77
USA West	AZ	65	40	64	37	40
	CA	159	128	149	189	161
	CO	49	62	56	99	41
	IL	32	58	55	86	76
	NV	31	31	47	39	14
	TX	213	191	202	237	209
	WA	117	84	92	76	62

In the same way, if you set the ‘Aggregate’ menu to ‘Max’, the summary value shown in the crosstab for the above row and column combination would represents the *maximum quantity* of ‘Business’ category products purchased by any ‘USA East’ region customer.

See Also

Crosstab Properties: Hierarchy Tab, for information on how to specify a custom drill-down hierarchy.

Aggregation Options

The table below lists the available crosstab aggregation measures. You can choose to display univariate aggregations (e.g., 'Sum', 'Count') as a *percentage value* by selecting the percentage basis (e.g., 'Group', 'GrandTotal') from the accompanying 'Percentage of' menu.

For the bivariate aggregation methods (e.g., 'Correlation', 'Weighted Average'), you will need to select a variable (column) to use as the second operand in the computation. Make this selection in the menu labeled 'with'.

Table 3. Aggregation Measures.

AGGREGATE MEASURE	DESCRIPTION
Sum	Displays the sum of the measure values for the given row and column headings
Average	Displays the average of the measure values for the given row and column headings
Max	Displays the maximum of the measure values for the given row and column headings
Min	Displays the minimum of the measure values for the given row and column headings
Count	Displays the total count of measure values for the given row and column headings. This represents the total number of records corresponding to the given row and column headings, and is the same value for any selected measure.
Distinct Count	Displays the count of unique measure values for the given row and column headings.
First	Displays the first value for the measure (for the given row and column headings) when sorted based on the values in a second column, specified by the menu labeled 'by'.
Last	Displays the last value for the measure (for the given row and column headings) when sorted based on the values in a second column, specified by the menu labeled 'by'.
Correlation	Displays the Pearson correlation coefficient for the correlation between the measure values (for the given row and column headings) and the corresponding values in a second column, specified by the menu labeled 'with'.
Covariance	Displays the covariance between the measure values (for the given row and column headings) and the corresponding values in a second column, specified by the menu labeled 'with'.
Variance	Displays the (sample) variance of the measure values for the given row and column headings.

AGGREGATE MEASURE	DESCRIPTION
Std Deviation	Displays the (sample) standard deviation of the measure values for the given row and column headings.
Variance (Pop)	Displays the (population) variance of the measure values for the given row and column headings.
Std Deviation (Pop)	Displays the (population) standard deviation of the measure values for the given row and column headings.
Weighted Average	Displays the weighted average of the measure values for the given row and column headings. The weights are given by the corresponding values in a second column, which is specified by the menu labeled 'with'.

See Also

Percentage Representation for Aggregation, to display crosstab data as a percentage.

3.3.4 Percentage Representation for Aggregation



You can display univariate aggregations (e.g., 'Sum', 'Count') as percentage values. To do this, select a basis for the percentage representation from the 'Percentage of' menu. Click the 'Options' button to select the dimension along which the percentage is computed (row or column).

The 'Percentage of' menu options are described in the table below

PERCENTAGE OPTION	DESCRIPTION
None	Displays the original aggregate measure values, with no percentages taken.
Group	Displays each aggregate measure value as a percentage of the aggregate value computed on the superordinate group (defined by the header one level above the finest-granularity header). This option is only available when two or more row headers or two or more column headers have been specified.
GrandTotal	Displays each aggregate measure value as a percentage of the aggregate value computed on the full row or column.

See Also

Crosstab Properties, for information on creating crosstabs.

3.3.5 Grouping Crosstab Headers into Named Groups

You can agglomerate data from different crosstab header values into larger categories called *named groups*. For example, you can combine column header labels designating individual states MA, CT, NY, NJ, etc., into larger groups such as ‘New England’ and ‘Mid-Atlantic’.

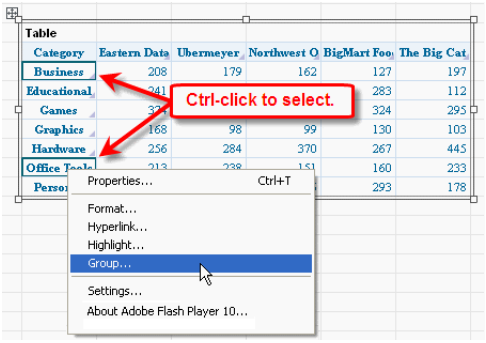
```
Named groups cannot be defined for date fields.
```

To create named groups for crosstab header labels, follow the steps below:

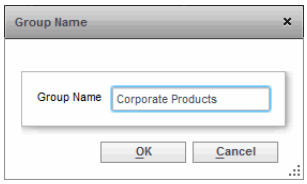
1. Ctrl-click to select the individual crosstab header labels (from the same heading level) that you want to combine into a single group.

You can group existing groups to make larger groups.

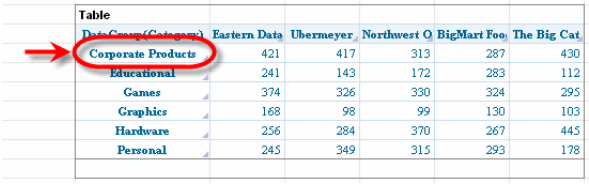
2. Right-click on one of the selected labels, and select ‘Group’ from the context menu. This opens the ‘Group Name’ dialog box.



3. Enter a name for the new group in the ‘Group Name’ dialog box, and click ‘OK’.

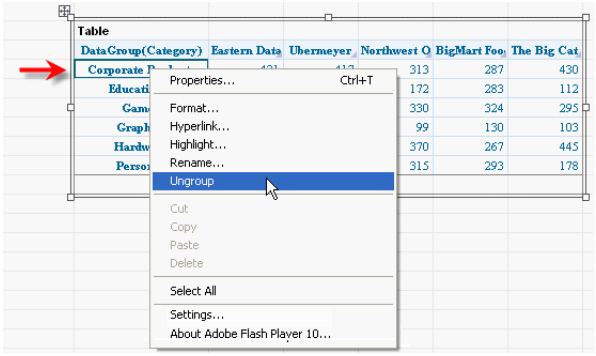


This replaces the selected labels with the label for the named grouping, and agglomerates the data corresponding to the original labels under the new label.



4. Repeat the above steps to add additional named groups.

To decompose an existing name group, right-click on the header label and select ‘Ungroup’ from the context menu.



See Also

Grouping Chart Labels into Named Groups, to create groupings in a chart.

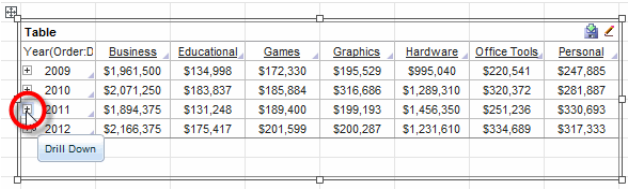
3.3.6 Drilling Down into a Crosstab

When a crosstab header displays a date field or a cube dimension, the crosstab automatically provides the user with the ability to drill down to finer levels of resolution of the date or dimension.

To drill down into a date or cube dimension on a crosstab header, follow these steps:

The ‘+’ symbol is only visible when you hover over the crosstab.

1. Hover the mouse over the crosstab. For headers that represent date or cube dimension data, this will display a ‘+’ symbol at the left side of the header.



2. Click the ‘+’ symbol to drill down to the next level of the date or dimension.

3. Repeat the above steps to drill down to deeper levels of the hierarchy.

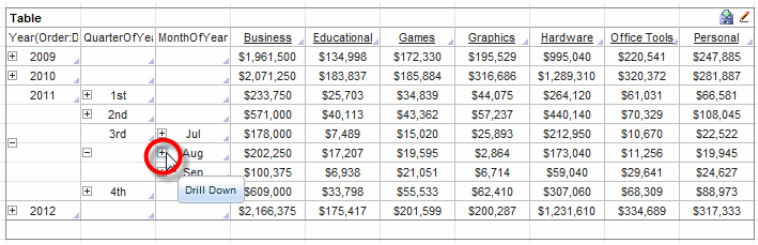


Table									
Year(Order:D	QuarterOfYe	MonthOfYear	Business	Educational	Games	Graphics	Hardware	Office Tools	Personal
2009			$1,961,500	$134,998	$172,330	$195,529	$995,040	$220,541	$247,885
2010			$2,071,250	$183,837	$185,884	$316,686	$1,289,310	$320,372	$281,887
2011	1st		$233,750	$25,703	$34,839	$44,075	$264,120	$61,031	$66,581
	2nd		$571,000	$40,113	$43,362	$57,237	$440,140	$70,329	$108,045
	3rd	Jul	$178,000	$7,489	$15,020	$25,893	$212,950	$10,670	$22,522
		Aug	$202,250	$17,207	$19,595	$2,864	$173,040	$11,256	$19,945
		Sep	$100,375	$6,938	$21,051	$6,714	$59,040	$29,641	$24,627
	4th		$609,000	$33,798	$55,533	$62,410	$307,060	$68,309	$88,973
2012			$2,166,375	$175,417	$201,599	$200,287	$1,231,610	$334,689	$317,333

4. To drill up to a higher level of the date or dimension hierarchy, hover the mouse over the crosstab. This displays a '–' symbol at the left side of expanded headers.

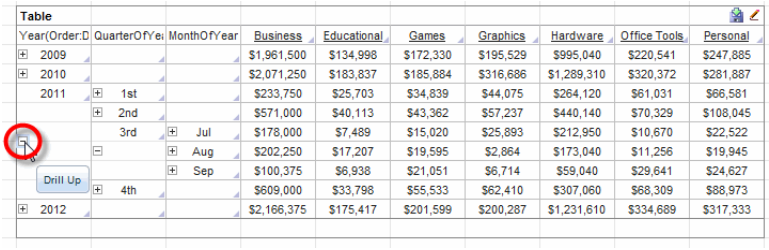


Table									
Year(Order:D	QuarterOfYe	MonthOfYear	Business	Educational	Games	Graphics	Hardware	Office Tools	Personal
2009			$1,961,500	$134,998	$172,330	$195,529	$995,040	$220,541	$247,885
2010			$2,071,250	$183,837	$185,884	$316,686	$1,289,310	$320,372	$281,887
2011	1st		$233,750	$25,703	$34,839	$44,075	$264,120	$61,031	$66,581
	2nd		$571,000	$40,113	$43,362	$57,237	$440,140	$70,329	$108,045
	3rd	Jul	$178,000	$7,489	$15,020	$25,893	$212,950	$10,670	$22,522
		Aug	$202,250	$17,207	$19,595	$2,864	$173,040	$11,256	$19,945
		Sep	$100,375	$6,938	$21,051	$6,714	$59,040	$29,641	$24,627
	4th		$609,000	$33,798	$55,533	$62,410	$307,060	$68,309	$88,973
2012			$2,166,375	$175,417	$201,599	$200,287	$1,231,610	$334,689	$317,333

5. Click the '–' symbol to drill up to the higher level of the date or dimension hierarchy.

To disable the drilling feature for a particular crosstab, deselect the 'Enable Drilling' option in the **Advanced** tab of the 'Crosstab Properties' dialog box. To specify a custom drill hierarchy, including custom date levels, see *Crosstab Properties: Hierarchy Tab*.

See Also

Crosstab Properties: Hierarchy Tab, for information on how to create a custom drill-down.

Drilling Down into a Chart, to perform the drill operation on a chart dimension.

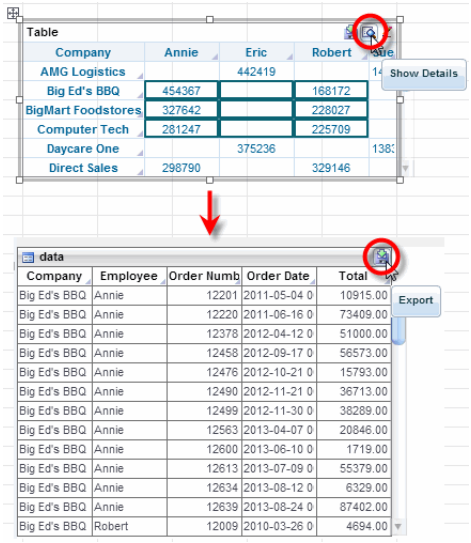
3.3.7 Exporting Crosstab Data

To export the aggregated data displayed in a Crosstab to an Excel file, see *Exporting Table Data for Viewing*.

To view or export the *detail* data on which the aggregations are based, follow the steps below:

1. Select the desired aggregates by clicking the appropriate cells, rows, or columns in the Crosstab.

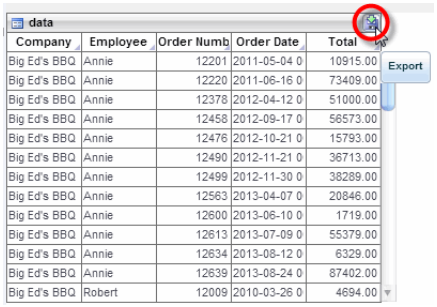
2. Click the 'Show Details' button at the top-right of the Crosstab. This opens a panel containing the detail data.



To close the data panel, click the red ‘X’ in the top-right corner.

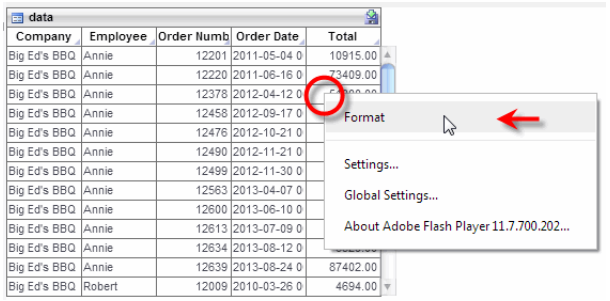
Exporting Detail Data

To export the detail data, press the ‘Export’ button in the data panel.

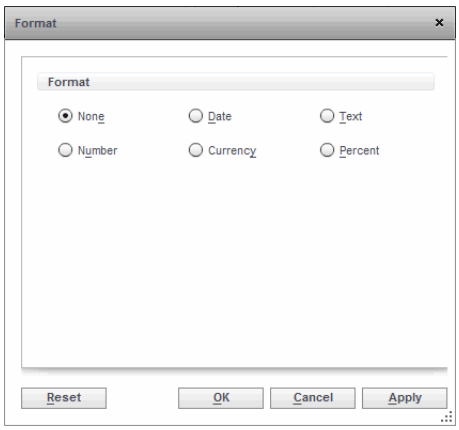


Formatting Detail Data

To format values in the data panel, right-click on the column you wish to format, and select ‘Format’ from the context menu.



This opens the ‘Format’ dialog box, which allows you to set the format for the column. See *Format Tab* for information about the available options.



Formatting that you specify is saved with the Viewsheet, and is automatically applied to the column whenever the ‘Show Details’ option is used, both in Visual Composer and the User Portal.

3.4 Charts

A chart displays a graphical view of summarized data (the graphical equivalent of a crosstab). Viewsheet charts are interactive. You can zoom in and out, view chart details, and “brush” the chart to track data in multiple views.

3.4.1 Creating a Chart

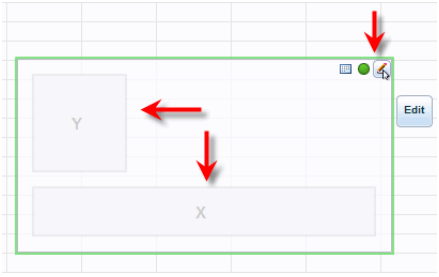
This section explains the basic procedure for creating and editing a chart, including binding the chart to data, setting ranking order, aggregation method, and chart style.

Creating a New Chart

To add a chart to a Viewsheet, drag and drop the Chart element from the component tree into the grid. Use the mouse to move and resize the chart as desired.

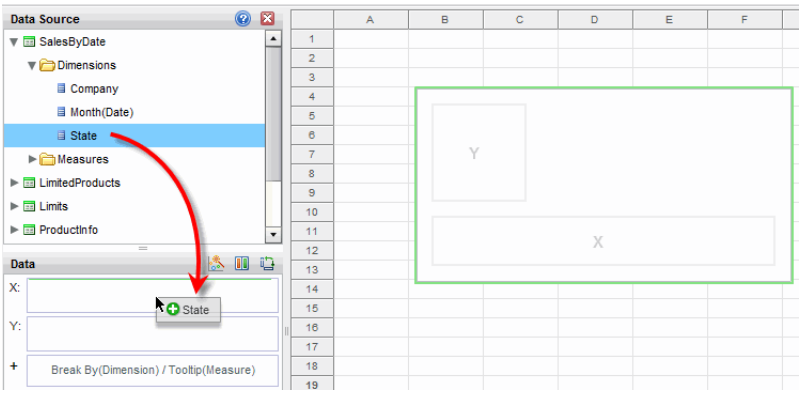
To bind the Chart to data, follow the steps below:

1. Click on the center of the chart, or click the ‘Edit’ button in the top-right corner. This opens the Chart Editor on the left side of the Visual Composer.



Tip: You can drag fields from the **Data Source** panel directly onto the Viewsheet grid to create a Selection component that filters the chart data.

2. From the **Data Source** panel in the Chart Editor, drag a dimension onto the 'X' field in the **Data** panel.

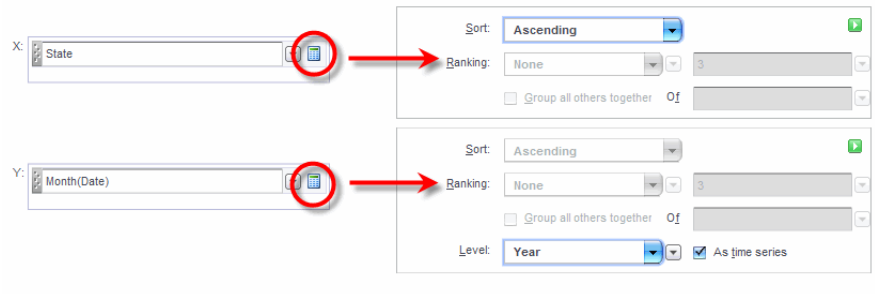


Note: When binding hierarchical cube dimensions, chart labels will include all hierarchical components. To avoid this, ask an administrator to set the property *olap.table.originalContent=false* in *sree.properties*.

This dimension provides the chart's X-axis data, and the dimension name is automatically used as the axis label.

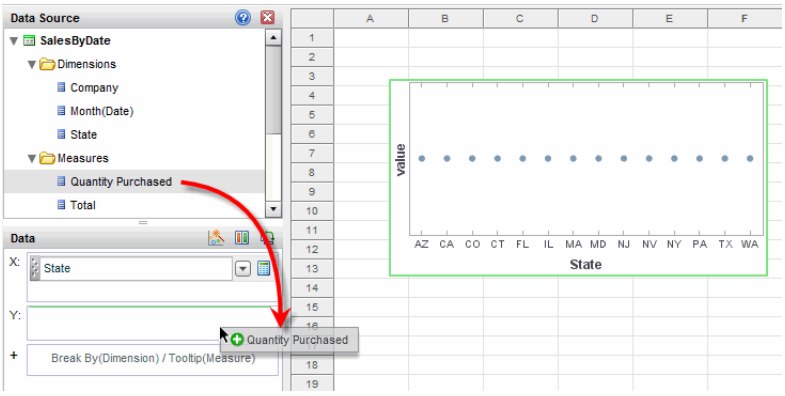
To rank a dimension based on a measure that is not displayed in the chart, see *Adding a Non-Displaying Measure to a Chart*.

3. Click the 'Edit Dimension' button, and select the desired ordering and ranking for the X-axis data.



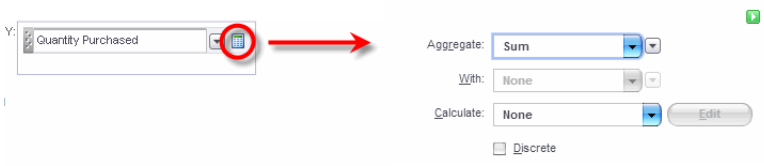
Tip: You can drag a measure field from the **Data Source** panel directly onto the Viewsheet grid to create a Range Slider component that filters the chart data.

4. From the **Data Source** panel, drag a measure onto the 'Y' field in the **Data** panel.



This measure provides the Chart's Y-axis data, and the measure name is automatically used as the axis label.

5. Click the 'Edit Measure' button, and select the desired aggregation method for the Y-axis data.



6. For a line-type chart or map-type chart (see *Setting the Chart Style*), optionally drag a field to the 'P' region. The values in this *path* field are used to assign the sort order for the plotted data. Connecting lines are drawn between points which are adjacent in the *path* sort order, rather than between points which are adjacent on the axis.

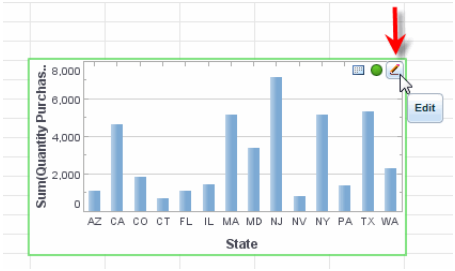
See the following sections for information about changing other aspects of the chart, including visual properties, labeling, and sub-series.

See Also

Creating a Map Chart, for information on how to create a map chart.
Editing a Dimension, for more information on ranking and ordering.
Editing Chart Formats, for information on changing chart labels.
Adding a Sub-Series, for information on using the **Visual** panel.
Creating Selection List from Chart Data Source, for information on adding a Selection List.
Creating a Selection Tree from Chart Data Source, for information on adding a Selection Tree.
Positioning a Component, for information on moving and resizing a chart.

Editing an Existing Chart

To edit an existing chart, click the 'Edit' button in the top-right corner of the Chart. This opens the chart editor on the left side of the Visual Composer.



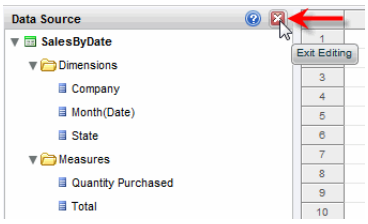
Note: If Ad Hoc editing is enabled, end-users can also access the Chart Editor to modify the chart.

A green outline designates the chart that is currently active in the editor.

Exiting the Chart Editor

To exit the Chart Editor, do one of the following:

- Click the Chart's 'Edit' button a second time.
- Right-click the Chart and select 'Exit Editing' from the context menu.
- Click the 'Exit Editing' button at the top of the Chart Editor.



See Also

Chart Properties: Advanced Tab, for information on enabling Ad Hoc editing.

Setting the Chart Style

To set the style of the chart (for all datasets on the chart), follow the steps below:

Note: The 'Auto' chart type automatically selects an appropriate chart type for the provided data.

1. In the **Data** panel of the Chart Editor, click the 'Select Chart Type' button.



2. Click a chart type to select it, and then click the ‘Apply’ button. (You can also double-click the desired chart type.)

To apply distinct styles to the different datasets on the chart, see *Setting a Chart Style for an Individual Dataset*.

Chart Examples

The following sections provide examples of how to create charts in different styles.

Creating a Pie Chart

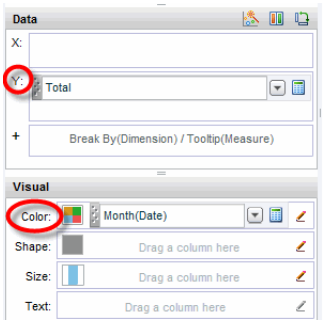
This example illustrates how to create a pie chart. Follow the steps below to create a pie chart that displays Total Sales according to Year.

1. Create a new Viewsheet based on the ‘Tutorial’ > ‘ProductSales’ Worksheet.

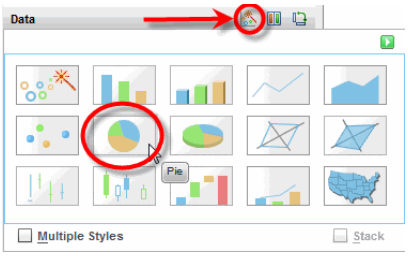
2. Drag a Chart component from the Component Tree to the Viewsheet, and click the center of the Chart to open the Chart Editor.

3. In the Chart Editor, drag the ‘Total’ field from the **Data Source** panel to the ‘Y’ region of the **Data** panel.

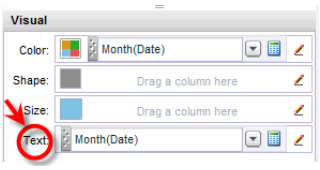
4. In the Chart Editor, drag the ‘Date’ field from the **Data Source** panel to the ‘Color’ region of the **Visual** panel.



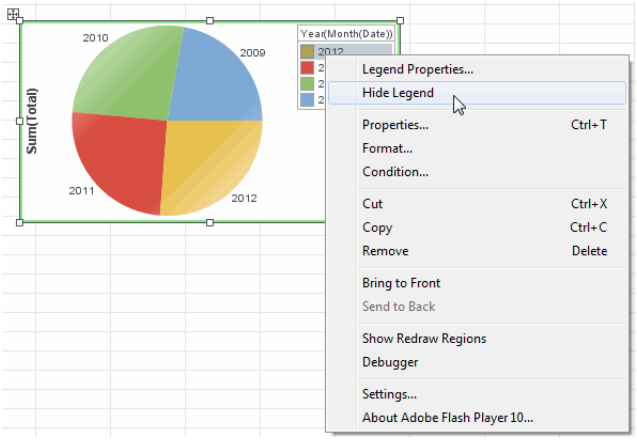
5. Click the ‘Select Chart Type’ button in the **Data** panel, and choose ‘Pie’. This converts the chart into a pie-chart representation.



6. Add the respective year values to the pie slices: Drag the ‘Date’ field from the **Data Source** panel to the ‘Text’ region of the **Visual** panel.



7. Remove the legend (if desired). Right-click on the legend, and select ‘Hide Legend’ from the context menu.



The pie chart is now complete. To place a small gap between the slices, set the ‘Explode Pie’ plot property. See *Editing Plot Properties* for more information.

¢

Creating a Candle Chart

A *candle chart* displays four different measures, “low,” “high,” “opening,” and “closing,” and is most often used to plot trading information. A candle chart is functionally similar to a *stock chart*.

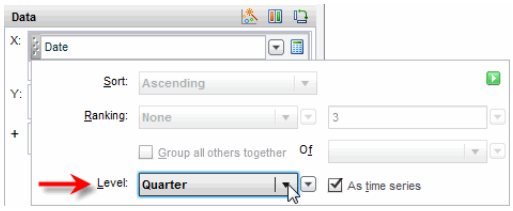
To create a candle chart that displays the averaged high, low, opening, and closing prices by quarter, follow the steps below:

1. Create a new Viewsheet based on the ‘Stock History’ > ‘Stock Prices’ query.

2. Add a new Chart component to the Viewsheet, and open the Chart Editor.

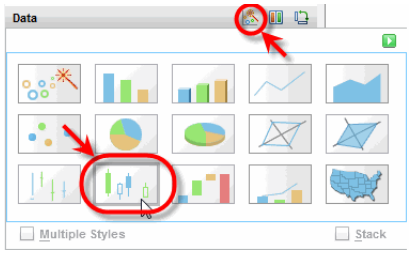
3. Drag the ‘Date’ field from the ‘Stock Prices’ data block to the ‘X’ region of the **Data** panel.

4. Press the ‘Edit Dimension’ button next to the ‘Date’ field, and choose ‘Quarter’ from the ‘Level’ menu. Then press the green ‘Apply’ button.

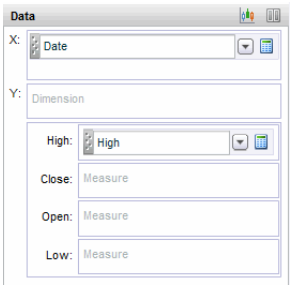


5. Drag the ‘High’ field of the ‘Stock Prices’ query to the ‘Y’ region of the **Data** panel.

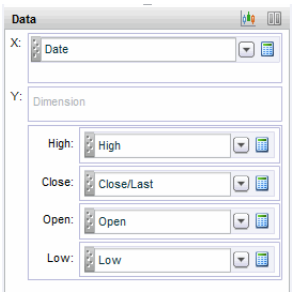
6. Press the ‘Select Chart Style’ button in the **Data** panel, and select the ‘Candle’ style. Press the green ‘Apply’ button.



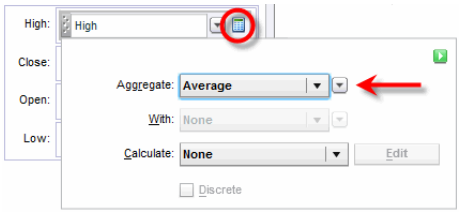
Note that the **Data** panel now provides binding regions for ‘High’, ‘Close’, ‘Open’, and ‘Low’.



7. Drag the ‘Close/Last’, ‘Open’, and ‘Low’ measures from the ‘Stock Prices’ data block to the appropriate regions of the **Data** panel, as shown below.

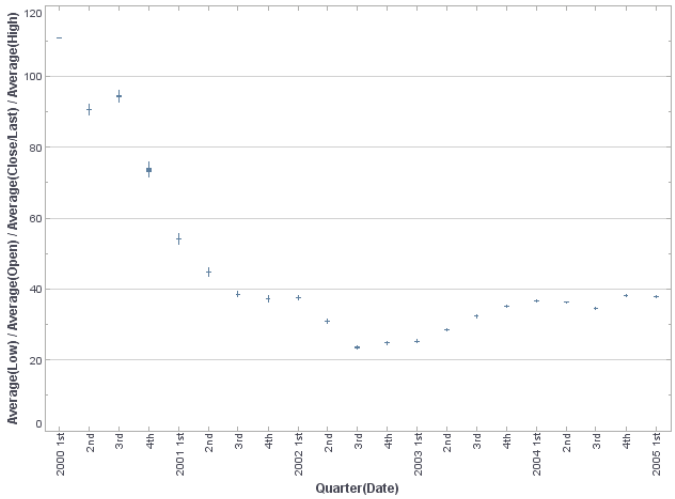


8. Press the ‘Edit Measure’ button next to the ‘High’ field, and choose ‘Average’ from the ‘Aggregate’ menu. Then press the green ‘Apply’ button.

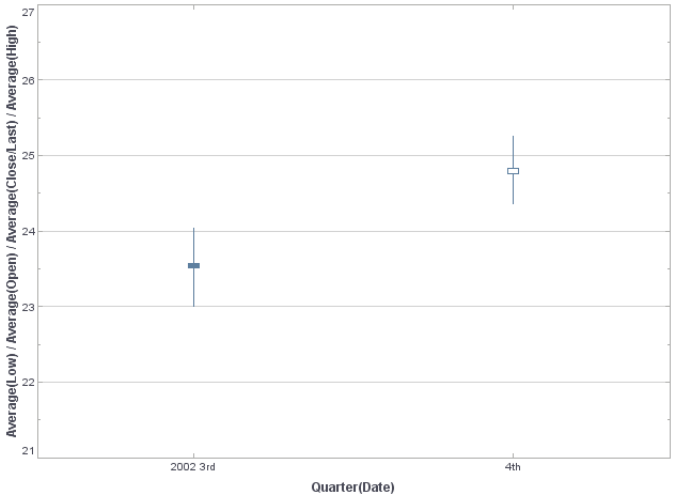


9. Repeat the above step to set the aggregate to ‘Average’ for the other measures: ‘Close/Last’, ‘Open’, and ‘Low’.

10. Preview the chart.



11. To observe the structure of the chart more closely, use the Zoom tool to view a more limited date range.



Note that for each quarter, the values of each of the measures (`High`, `Low`, `Close/Last`, `Open`) have been independently averaged. The “high” and “low” measures are represented by the extremes of the candle “wick,” and the “open” and “close” measures are represented by the extremes of the candle “body.” Candles for which the “open” measure exceeds the value of the “close” measure are shown filled. Candles for which the “close” measure exceeds the value of the “open” measure are shown unfilled.

¢

See Also

Setting a Chart Style for an Individual Dataset, to set dataset-specific styles.

Creating a Table-Chart Hybrid

This example illustrates how to create a chart that has features of a table, in particular, a columnar display of data. Follow the steps below:

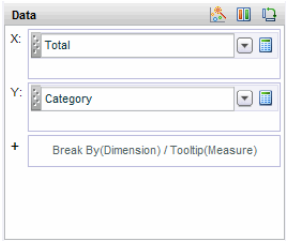
1. Create a new Viewsheet based on the 'Sales' > 'Sales Explore' Worksheet.

2. Drag a Chart from the **Component** panel into the Viewsheet grid, and resize the Chart to be wider and taller.

3. Click the 'Edit' button on the Chart to open the Chart Editor.

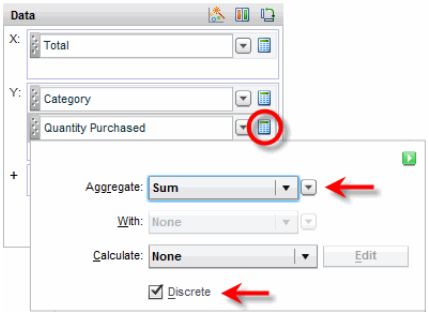
4. In the Chart Editor, drag the 'Total' field from the 'Measures' folder in the **Data Source** panel to the 'X' region of the **Data** panel.

5. Drag the 'Category' field from the 'Dimensions' folder of the **Data Source** panel to the 'Y' region of the **Data** panel.



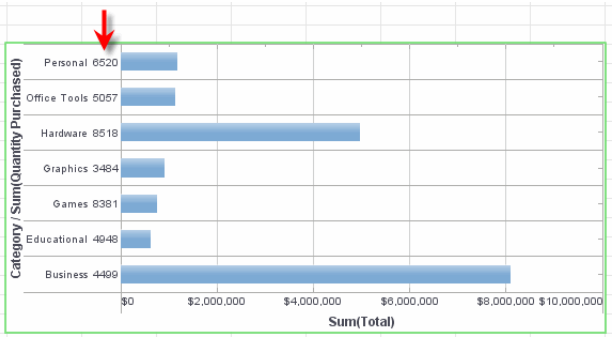
6. Drag the 'Quantity Purchase' field from the 'Measurers' folder in the **Data Source** panel to the 'Y' region of the **Data** panel (under 'Category').

7. Press the 'Edit Measure' button next to 'Quantity Purchased'. Set the 'Aggregate' property to 'Sum' and enable the 'Discrete' option. This allows the measure to behave as a dimension.



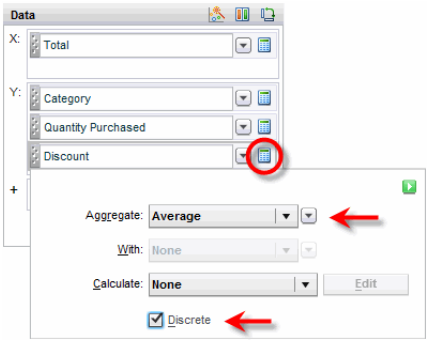
8. Press the green 'Apply' button.

Observe that the chart now displays a column of values giving the aggregated ‘Quantity Purchased’ value for each ‘Category’ group.



9. Drag the ‘Discount’ field from the ‘Measures’ folder in the **Data Source** panel to the ‘Y’ region of the **Data** panel (under ‘Quantity Purchased’).

10. Press the ‘Edit Measure’ button next to ‘Discount’. Set the ‘Aggregate’ property to ‘Average’ and enable the ‘Discrete’ option. This again allows the measure to behave as a dimension

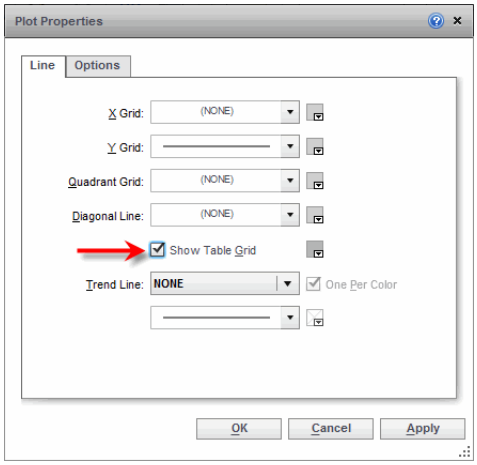


11. Press the green ‘Apply’ button.

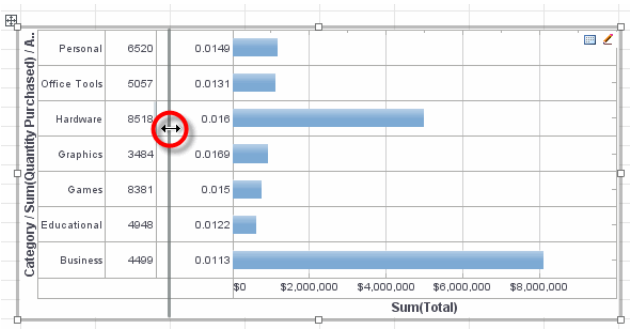
    Observe that the chart now displays a column of values giving the aggregated ‘Discount’ value for each ‘Category’ group.

12. Right-click in the plot area and select ‘Plot Properties’ from the context menu. This opens the ‘Plot Properties’ dialog box.

13. Enable the ‘Show Table Grid’ option, and press ‘OK’. This displays grid lines around the plot area, and around each text column.



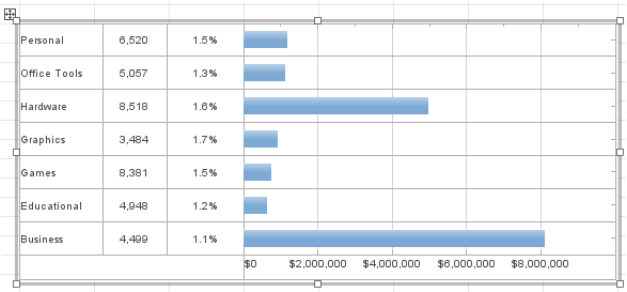
14. Adjust the spacing between the columns by dragging on the boundaries between columns.



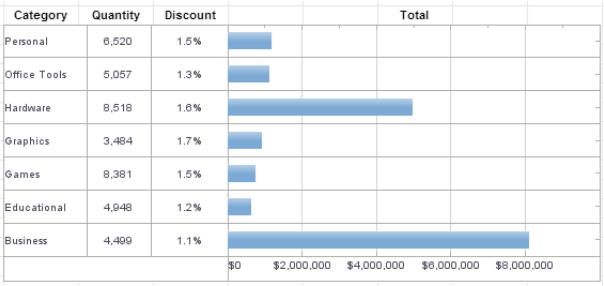
15. Click on each column of text in turn and set the desired formatting. (You can do this by using the toolbar buttons or by using the 'Format' dialog box. See *Format Dialog Box* for more information.)

    For example, set a number format for the 'Quantity Purchased' column and set a percent format for the 'Discount' column. Center-align both of these columns, and left-align the 'Category' column.

16. Right-click the X-axis title, and select 'Hide Title' from the context menu. Do the same for the Y-axis title.



17. Resize the chart as desired. Add text elements above the chart to provide any desired column headers.



Creating a Dot Plot

A *dot plot* represents a measure by its individual data points (without aggregation). This type of chart can be useful when the dataset is very small, with fewer than approximately 20 data points for each dimension value.

```
Note: If the dataset contains too many records, the chart will not
be able to display all of the data points.
```

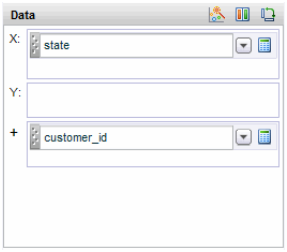
In this example, you will create a dot plot that displays the companies in each state. Follow the steps below:

1. Create a new Viewsheet based on the ‘customers’ query (under ‘Data Source’ > ‘Orders’).

2. Drag a Chart from the **Component** panel into the Viewsheet grid. Press the ‘Edit’ button on the Chart to open the Chart Editor.

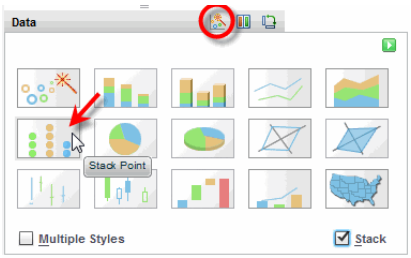
3. In the Chart Editor, drag the ‘state’ field from the ‘Dimensions’ folder in the **Data Source** panel to the ‘X’ region of the **Data** panel.

4. Drag the ‘customer_id’ field from the ‘Measures’ folder of the **Data Source** panel to the ‘Break By’ region of the **Data** panel.

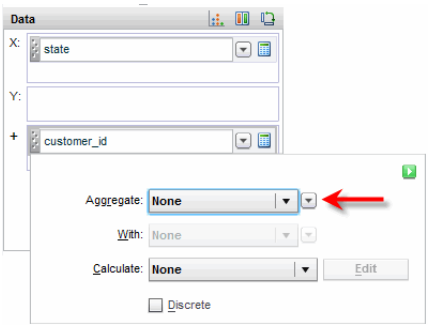


Alternatively, you can drag the ‘customer_id’ field to the ‘Color’, ‘Shape’, or ‘Size’ regions of the **Visual** panel if you wish to distinguish the data points using a visual format.

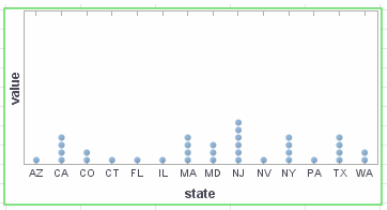
5. Press the ‘Select Chart Style’ button in the **Data** panel. Select the ‘Point’ type chart, and enable the ‘Stack’ option. Press the green ‘Apply’ button.



6. Press the ‘Edit Measure’ button next to ‘customer_id’. Set the ‘Aggregate’ property to ‘None’, and press the green ‘Apply’ button.

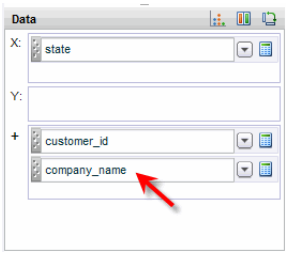


You have now created a basic dot plot.

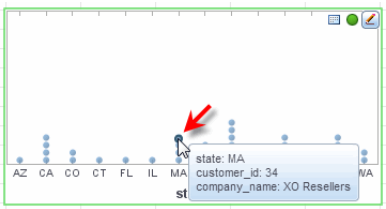


7. (Optional) Right-click on the Y-axis title and select ‘Hide Title’ from the context menu.

8. (Optional) Drag the ‘company_name’ field from the ‘Dimensions’ folder of the **Data Source** panel to the ‘Break By’ region of the **Data** panel. Place it under the existing ‘customer_id’ field.



This allows the chart tooltip to display both the ‘customer_id’ and ‘company_name’ values.

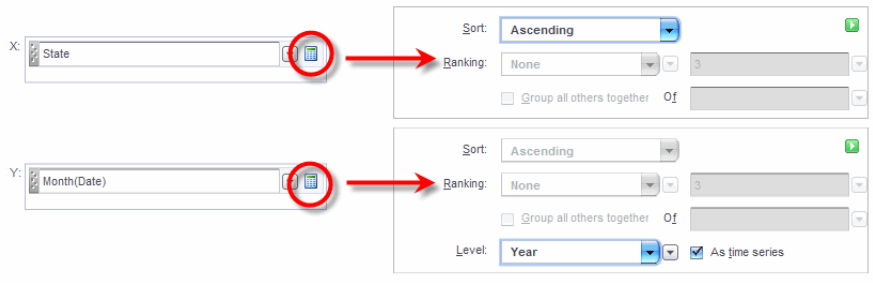


To further customize the tooltip, see *Custom Tooltip*.

3.4.2 Editing a Dimension

The Chart Editor automatically categorizes non-numerical fields of a Data Block as “dimensions,” unless otherwise specified. Typically, dimensions are represented on the X-axis or expressed as a subseries (i.e., ‘Visual’ binding). Dimensions can also be used in the ‘Break By (Dimension) / Tooltip (Measure)’ field of the **Data** panel.

To edit the representation of a dimension, click the ‘Edit Dimension’ button in the **Data** panel. This opens the options panel.

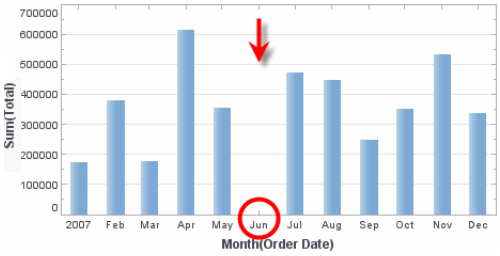


Note: A measure must be defined in order to rank or sort by value.

The following settings are available. Note that a field of type Date has additional grouping level options.

Sort	Change the ordering of the X-axis labels: **None**: Use the label order as defined within the data block or cube. **Ascending**: Place the labels in alphabetically ascending order. **Descending**: Place the labels alphabetically descending order. **By Value(Asc)**: Place the labels in ascending order based on the selected measure. **By Value(Desc)**: Place the labels in descending order based on the selected measure. **Manual**: Click the 'Edit' button to open the 'Manual Ordering' dialog box, and manually reorder the X-axis labels.
Ranking	Filter the axis labels by aggregate measure value. **None**: No ranking of X-axis labels. **Top**: Select the top N X-axis labels based on the selected measure. **Bottom**: Select the bottom N X-axis labels based on the selected measure.
Group all others together	For Top/Bottom-N ranking, creates an "Others" category which represents (in aggregate) all the groups excluded by the Top/ Bottom-N filter.
Level	Set the grouping level for dates. Note the distinction between options such as 'Month' and 'Month of Year': The 'Month' option groups each date according to composite month and year, e.g., Jan'07, Feb'07, ..., Dec'07, Jan'08, Feb'08, ..., Dec'08. The 'Month of Year' option groups each date strictly on the month component, e.g., Jan, Feb, ..., Dec, so that each group may include data from multiple years. Likewise, for the other 'of Year/of Month/of Week/ of Day' options. If the date field is already grouped by the database, select 'None'.
As time series	Specifies that gaps in Date data should be retained. For example, if the data in the chart is grouped by month, and there is no data for the month of June, the 'As time series' property ensures that the month of June is still retained in the chart. To change the way that the gap is displayed, see *Editing Plot Properties*.

The image below illustrates the effect of enabling the 'As time series' option.



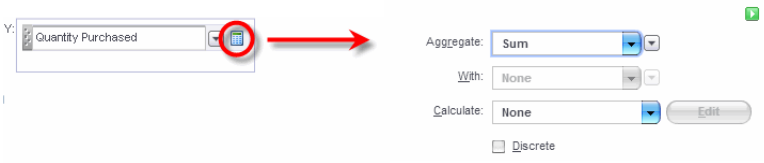
See Also

Sorting a Chart Dimension by Value, for a quick way to sort chart groups by value.

3.4.3 Editing a Measure

The Chart Editor automatically categorizes numerical fields of a Data Block as "measures." Typically, measures are represented on the Y-axis or expressed using one of the 'Visual' bindings. Measures can also be used in the 'Break By (Dimension) / Tooltip (Measure)' field of the **Data** panel.

To edit the representation of a measure, click the 'Edit Measure' button in the **Data** panel. This allows you to change the method of aggregation and calculation.



The following settings are available:

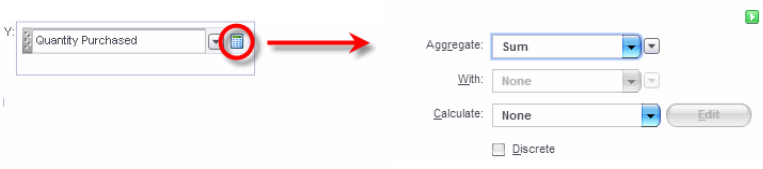
Aggregate	The method by which measure data should be aggregated. See *Aggregation Options* for a description of the methods.
With	For bivariate aggregations (such as 'Covariance' and 'Correlation'), this specifies the second operand.
Calculate	The computation that determines how values are represented on the chart (i.e., moving average, percentage, difference, etc.). See *Calculating a Measure Representation*.

Calculating a Measure Representation

There are many different ways to represent an aggregated measure on a chart. For example, you can represent the aggregated measure (for an X-axis group or subseries group) in terms of its percentage of a total, or by its difference from a preceding group, or by a moving average based on adjacent groups, etc.

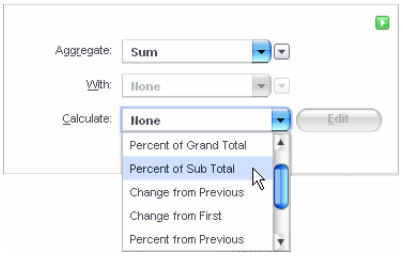
To use one of these representations, set the appropriate *calculation* method. Follow the steps below:

1. Click the 'Edit Measure' button next to a measure (either in the **Data** panel or **Visual** panel). This opens a pop-up panel.



Select 'None' (default) to use the raw aggregate. Select 'Custom' to create a new calculation method.

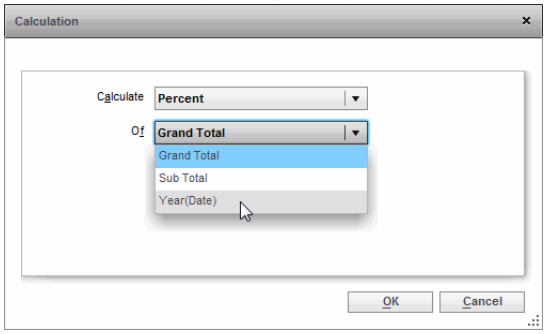
2. Select one of the calculation methods from the 'Calculate' menu in the pop-up panel.



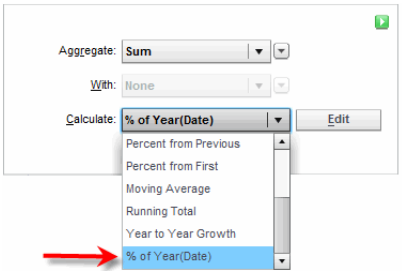
3. Click the 'Edit' button. This opens the 'Calculation' dialog box, which allows you to modify the properties of the calculation method. To do this, follow the steps below:

   a. From the 'Calculate' menu, select a type of calculation: 'Percent', 'Change', 'Running', or 'Sliding'.

   b. Select any additional options to fully specify the calculation method. (The options vary for the different methods. See *Calculation Methods* for more detail.)



   c. Click 'OK' to close the dialog box. This adds the new method to the 'Calculate' menu.



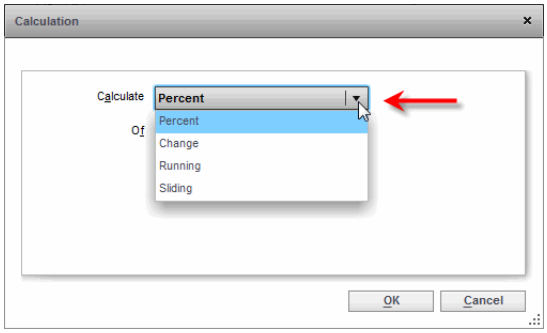
4. Click the green 'Apply' button.

The aggregated measure will now be displayed on the chart using the particular representation that you have specified.

See Also
Calculation Methods, for details on configuring the various methods.
Charts, for step-by-step instructions on creating a chart.

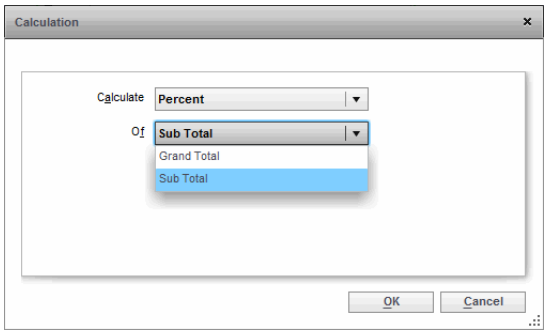
Calculation Methods

There are three basic types of calculations: 'Percent', 'Change', 'Running', and 'Sliding'. These methods are described in greater detail below.



Percent Calculation

The 'Percent' calculation allows you to express the measure based on 'Grand Total', 'Subtotal', or a particular dimensional group.



- **Dimension**: Expresses the value of the aggregated measure for each group as a percentage of the measure aggregated across the selected dimension.
- **Grand Total**: Expresses the value of the aggregated measure for each group as a percentage of the measure aggregated across all groups.
- **Subtotal**: For a facet-type chart (i.e., a chart with both a measure and dimension on the same axis or with multiple dimensions on the same axis), this expresses the value of the aggregated measure for each group as a percentage of the measure aggregated across all groups on the same sub-chart.

Example: Percent Calculation

In this example, you will create a chart that displays the quantity of products sold for different categories, regions, and states. You will display the aggregated measure for each subgroup as a percentage of the total aggregate for the category. Follow the steps below:

1. Create a new Viewsheet based on the sample 'Sales Explore' Worksheet in the 'Sales' folder.

2. Drag a Chart component onto the Viewsheet grid.

3. Click the center of the Chart (or click the 'Edit' button at the top-right). This opens the Chart Editor.

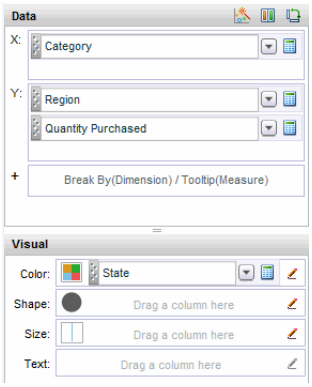
4. In the Chart Editor, make the following selections:

   a. From the 'Dimensions' node in the **Data Source** panel, drag the 'Category' dimension to the 'X' field in the **Data** panel.

   b. From the 'Measures' node in the **Data Source** panel, drag the 'Quantity Purchased' measure to the 'Y' field in the **Data** panel.

   c. From the 'Dimensions' node in the **Data Source** panel, drag the 'Region' dimension to the 'Y' field in the **Data** panel, and drop it above the 'Quantity Purchased' measure. (This creates a *facet* chart containing multiple sub-graphs.)

   d. From the 'Dimensions' node in the **Data Source** panel, drag the 'State' dimension to the 'Color' field in the **Visual** panel. This creates a subseries based on the 'State' field.



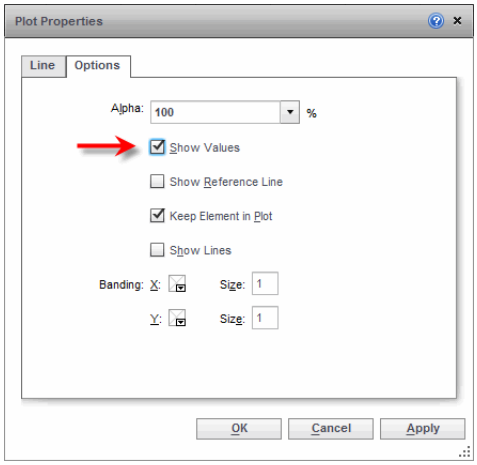
5. Drag the Chart handles to enlarge the Chart as desired.

6. From the ‘Dimensions’ node in the ‘Data Source’ tree, drag the ‘State’ dimension onto the Viewsheet grid. This creates a ‘State’ Selection List.

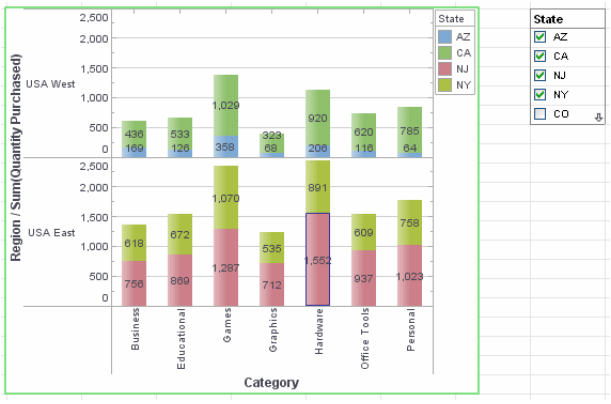
7. Select the following states in the ‘State’ selection list: ‘AZ’, ‘CA’, ‘NJ’, ‘NY’.

8. Right-click the plot area of the chart, and select ‘Plot Properties’ from the context menu. This opens the ‘Plot Properties’ dialog box.

9. Enable the ‘Show Values’ option, and click ‘OK’.

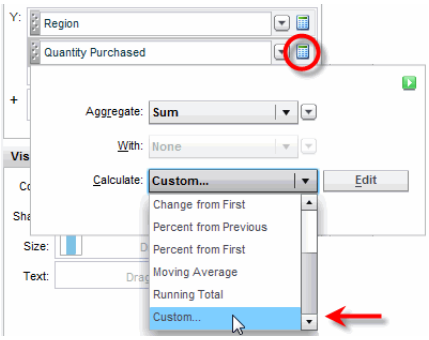


This displays the aggregated measure value for each group on the chart.



10. In the **Data** panel of the Chart Editor, click the 'Edit Measure' button next to the 'Quantity Purchased' measure. This opens a pop-up panel.

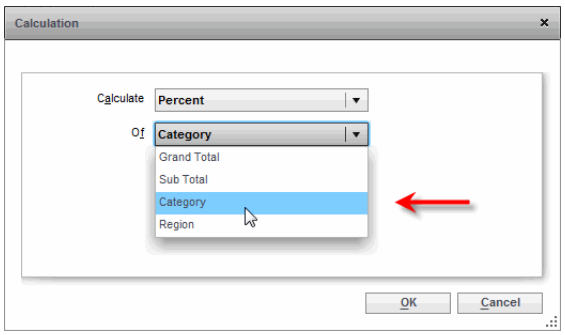
11. In the 'Calculate' menu of the pop-up panel, select the 'Custom' option.



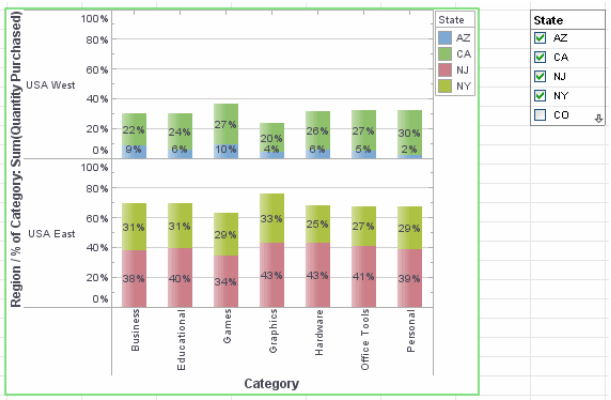
12. Click the 'Edit' button to open the 'Chart Calculation' dialog box.

13. In the 'Chart Calculation' dialog box, select 'Percent' from the 'Calculate' menu.

14. Select 'Category' from the 'Of' menu, and click 'OK' to close the dialog box.



15. Click the green ‘Apply’ button in the pop-up panel to apply the change.



The aggregated measure for each group is now represented as its percentage of the total aggregate for the corresponding category.

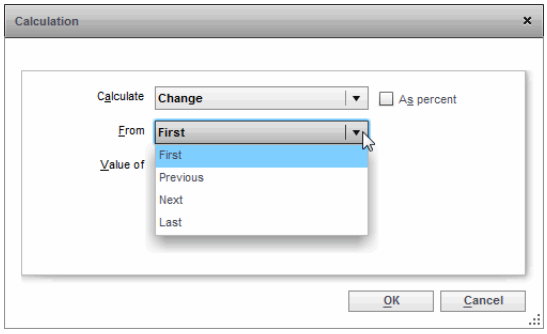
Experiment with the other available representations for the group aggregates:

- **Grand Total**: The aggregated measure for each group is represented as its percentage with respect to the entirety of chart data.
- **Subtotal**: The aggregated measure for each group is represented as its percentage with respect to the entirety of data on the same sub-graph (i.e., the region, ‘USA East’ or ‘USA West’).
- **Region**: The aggregated measure for each group is represented as its percentage with respect to its entire region (same as ‘Subtotal’ for this example).
- **State**: The aggregated measure for each group is represented as its percentage with respect to its entire state.

¢

Change Calculation

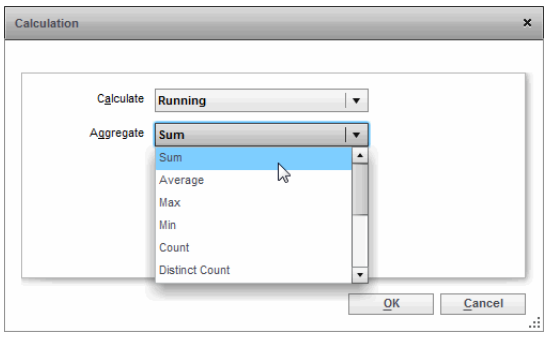
The ‘Change’ calculation allows you to express the group aggregate in terms of its deviation (or percent deviation, if ‘As percent’ is selected) from the preceding, succeeding, first, or last group in the series. The ‘From’ menu specifies the baseline value.



- **First**: Expresses the value of the aggregated measure for each group as a difference from the corresponding first value for the parent group selected in the 'Value of' menu.
- **Previous**: Expresses the value of the aggregated measure for each group as a difference from the corresponding previous value for the parent group selected in the 'Value of' menu.
- **Next**: Expresses the value of the aggregated measure for each group as a difference from the corresponding next value for the parent group selected in the 'Value of' menu.
- **Last**: Expresses the value of the aggregated measure for each group as a difference from the corresponding last value for the parent group selected in the 'Value of' menu.

Running Calculation

The 'Running' calculation allows you to express each group aggregate as an accumulation of previous aggregate values in the series. The method of accumulation is specified by the 'Aggregate' menu in the 'Calculation' dialog box. (See *Aggregation Options* for more about the different methods.)



The 'Reset at' option, available for date fields, allows you to specify the date interval (e.g., year, quarter, week, etc.) at which the accumulation should be cleared.

Example:
Running Average

In this example, you will create a chart that computes total quantities sold by month, and displays these values as a running average. The running average will be reset on a yearly basis. Follow the steps below:

1. Create a new Viewsheet based on the sample ‘Sales Explore’ Worksheet in the ‘Sales’ folder.

2. Drag a Chart component onto the Viewsheet grid, and enlarge the Chart as desired by dragging the handles.

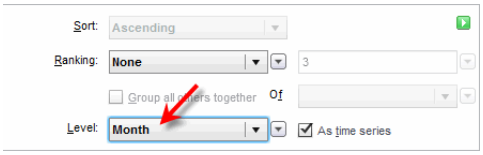
3. Click the center of the Chart (or click the ‘Edit’ button at the top-right). This opens the Chart Editor.

4. In the Chart Editor, from the ‘Dimensions’ node in the **Data Source** panel, drag the ‘Date’ dimension to the ‘X’ field in the **Data** panel.

5. From the ‘Measures’ node in the **Data Source** panel, drag the ‘Quantity Purchased’ measure to the ‘Y’ field in the **Data** panel.

6. In the **Data** panel of the Chart Editor, click the ‘Edit Dimension’ button next to the ‘Date’ measure. This opens a pop-up panel.

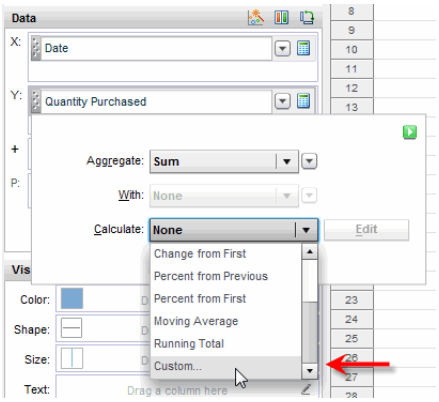
7. From the ‘Level’ menu select ‘Month’, and click the green ‘Apply’ button. This groups the date data (i.e., X-axis labels) by month.



8. In the **Data** panel of the Chart Editor, click the ‘Edit Measure’ button next to the ‘Quantity Purchased’ measure. This opens a pop-up panel.

Note: Leave the ‘Aggregate’ option in the pop-up panel set to the default ‘Sum’.

9. In the ‘Calculate’ menu of the pop-up panel, select the ‘Custom’ option.

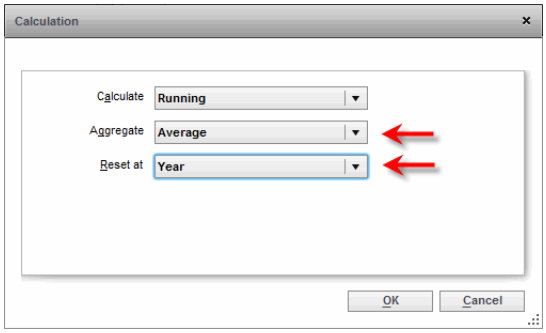


10. Click the ‘Edit’ button to open the ‘Chart Calculation’ dialog box. Make the following settings:

    a. From the ‘Calculate’ menu, select ‘Running’.

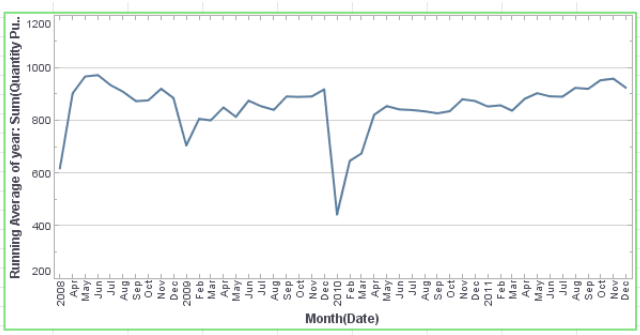
    b. From the ‘Aggregate’ menu, select ‘Average’.

    c. From the ‘Reset at’ menu, select ‘Year’.



    d. Click ‘OK’ to close the dialog box.

11. Click the green ‘Apply’ button in the pop-up panel to apply the change.



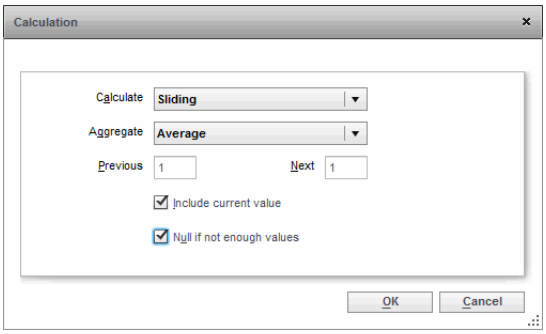
The time series now shows a running average for the summed quantity purchase, re-initialized at each new year.

¢

See Also
Sliding Calculation, to accumulate values over a limited sliding range.

Sliding Calculation

The ‘Sliding’ calculation allows you to express each group’s value as an accumulation of neighboring aggregate values in the series, specified by a rectangular sliding window. This generally has the effect of smoothing (low-pass filtering) the displayed data.



The method of accumulation is specified by the ‘Aggregate’ menu in the ‘Calculation’ dialog box. (See *Aggregation Options* for more information about the different methods.) The ‘Previous’ and ‘Next’ values specify the span of the sliding window as the number of preceding and succeeding groups, respectively, to include in the calculation. All included groups have equal weight in the calculation.

The ‘Include current value’ options incorporates each group’s aggregate value into its own calculation. When this option is not enabled, the calculation for a group uses only its neighboring groups’ values. For example, assume the chart displays sales totals for March, April, May, June, and July, and the sliding window is 3 units wide (‘Previous’=1 and ‘Next’ =1). If ‘Include current value’ is enabled, the displayed value for May is aggregated from three months’ data, April, May, and June. However, if ‘Include current value’ is not enabled, the displayed value for May is aggregated from two months’ data, April and June.

The ‘Null if not enough values’ option suppresses chart points for which the sliding window does not obtain the required span. For example, assume the chart displays sales totals for March, April, May, June, July, August, and September, and that the sliding window is 5 units wide (‘Previous’=2 and ‘Next’ =2). If ‘Null if not enough values’ is enabled, then no points are

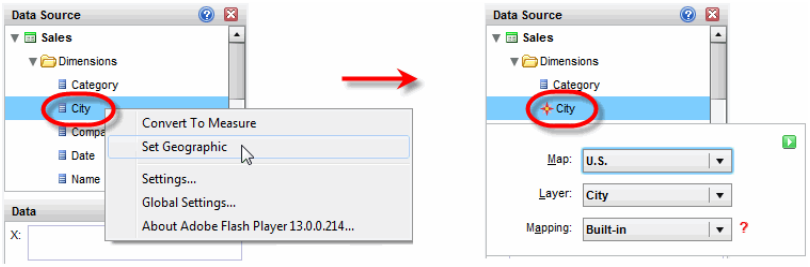
displayed on the chart for the months of March and April because the calculation cannot be performed for these groups as specified (there is no data available preceding March.) However, if 'Include current value' is not enabled, the aggregates for March and April will be computed and displayed using the data available within the sliding window.

See Also
Running Calculation, to accumulate values from the start of the series.

Setting a Geographical Field

To set a field as a geographical field (for use with *Map Charts*), follow the steps below:

1. Right-click the field in the tree, and select 'Set Geographic' from the context menu. This places a "star" icon next to the field name, and opens the geographical level editor.



The map that you select for a geographic field applies to all other geographical fields.

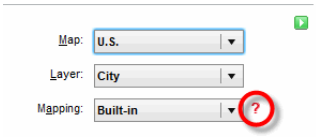
2. In the level editor, select the desired region from the 'Map' menu: Asia, Canada, Europe, Mexico, U.S., or World.

3. From the 'Layer' menu, select the geographical layer that corresponds to the data in the selected field.

   This specifies the layer of the geographical database against which field values should be matched (e.g., city names should be matched against the city layer, etc.). The options available in the 'Layer' menu depend on the previously selected 'Map' option.

4. From the 'Mapping' menu, select the desired map database.

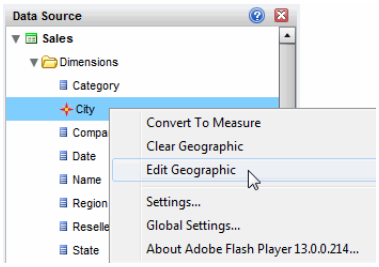
   The Chart Editor attempts to resolve values in the selected geographic field against the corresponding layer in the map database. If all data values are successfully matched, a green check mark is displayed. If some data values cannot be matched, a red question mark is displayed.



5. To resolve any unmatched names, click the red question mark. This opens the 'Geographical Mapping' dialog box. See *Resolving Geographical Data* for further instructions.

6. Click the 'Apply' button.

To modify the settings for an existing geographical field, right click the field, and select 'Edit Geographic' from the context menu. This opens the geographical level editor described above.



To remove the geographical status of a field, right click the field, and select 'Clear Geographic' from the context menu.

See Also

Resolving Geographical Data, for information on specifying a new mapping.

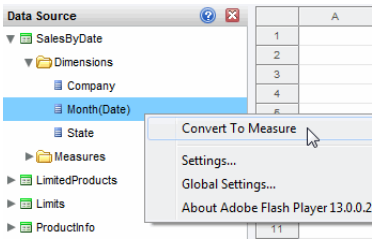
Map Charts, for information about representing geographical data on a map.

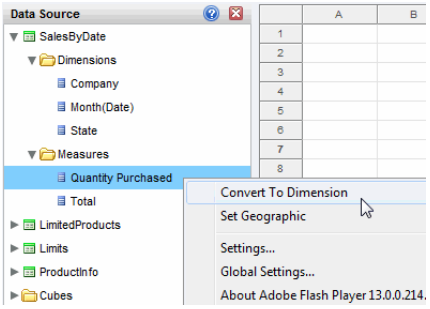
Converting between Measure and Dimension

By default, dimensions are displayed as discrete values, whereas measures are aggregated and displayed as a continuous range.

To convert a dimension to a measure, right-click the dimension in the tree, and select 'Convert to Measure' from the context menu. The new measure is automatically moved to the correct branch of the tree.

To convert a measure to a dimension, right-click the measure in the tree, and select 'Convert to Dimension' from the context menu.The new dimension is automatically moved to the correct branch of the tree.





To specify that a particular measure be displayed using discrete values (rather than a continuous spectrum), see *Displaying a Measure as Discrete Values*.

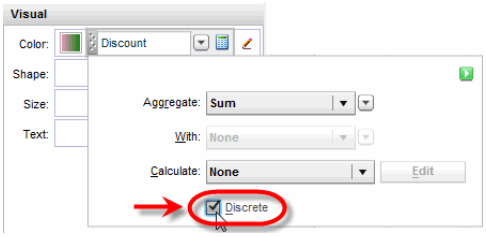
Displaying a Measure as Discrete Values

By default, dimensions are displayed as discrete values, and measures are displayed as a continuous range.

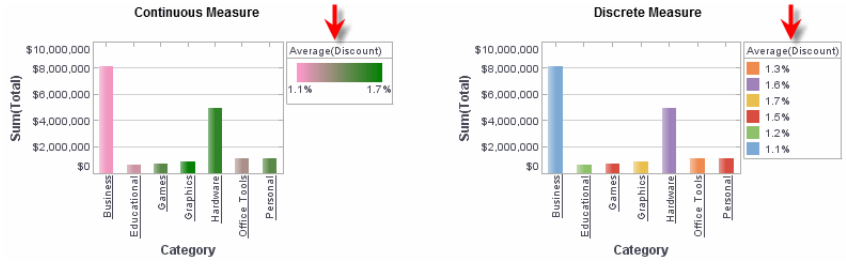
To force a measure to display as discrete values, follow the steps below:

1. Press the ‘Edit Measure’ button next to the measure name. This opens the ‘Edit Measure’ panel.

2. Enable the ‘Discrete’ option, and press the green ‘Apply’ button.



The measure will now display as discrete values rather than as a continuous spectrum.



To convert a measure to a dimension, see *Converting between Measure and Dimension*.

3.4.4 Editing Chart Formats

You can alter almost every aspect of a chart's display. The following sections explain how to modify titles, axes, legends, and the plot area.

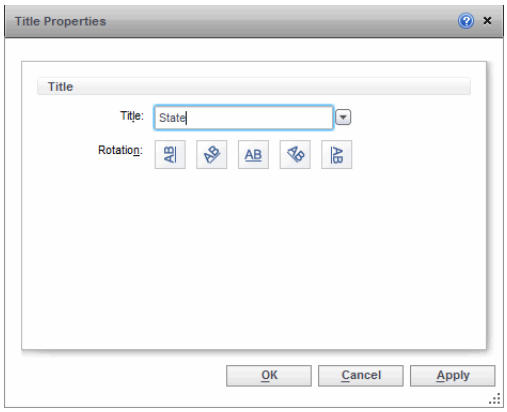
See Also
Chart Properties, for information about global chart properties.

Editing an Axis Title

Viewsheet Charts do not posses a "chart title." To add a chart title, use a text component.

To edit the text and rotation of an axis title, follow the steps below:

1. Right-click the title text on the Chart, and select 'Title Properties' from the context menu. This opens the 'Title Properties' dialog box.



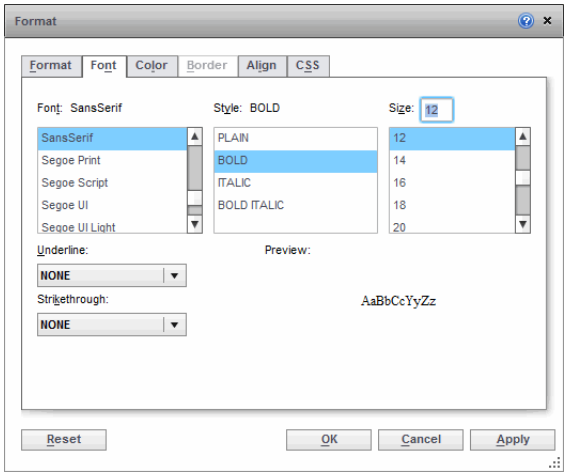
2. In the 'Title' field, enter the title for the axis. (The title text you enter will persist even if you later change the axis data.)

   To add the default axis name into your custom title, insert a placeholder such as {0} in the title. For example, if the chart displays two measures on the y-axis, 'Total' and 'Quantity', you can enter a custom title such as "{0} and {1} for all Employees" to display the following y-axis title: "Sum(Total) and Sum(Quantity) for all Employees".

3. Press 'OK' to close the dialog box and apply the changes.

To change the formatting of an axis title, follow the steps below:

1. Right-click the title text on the Chart, and select 'Format' from the context menu. This opens the 'Title Properties' dialog box.



2. Select a font, color, alignment, etc., for the axis title. See *Format Dialog Box* for more information about the options.

3. Click ‘OK’ to close the dialog box and apply the changes.

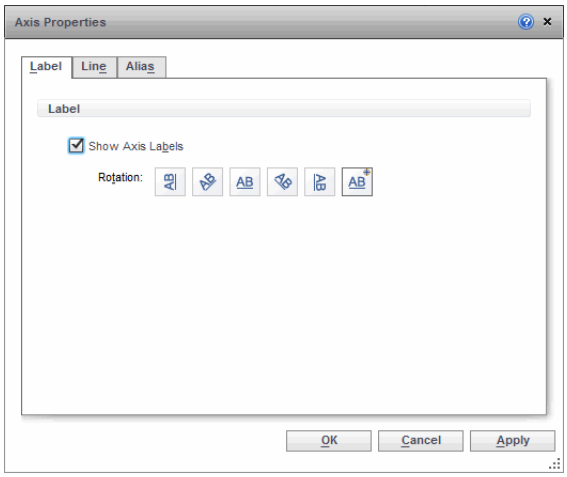
Editing a Chart Axis

To edit the labels and line format of an axis, right-click the axis or axis label, and select ‘Axis Property’ from the context menu. This opens the ‘Axis Properties’ dialog box.

```
Note: To hide axis line and labels, right-click the axis and choose
‘Hide Axis’ from the context menu. To reveal a hidden axis, right-
click the chart and select ‘Show All Axes’ from the context menu.
```

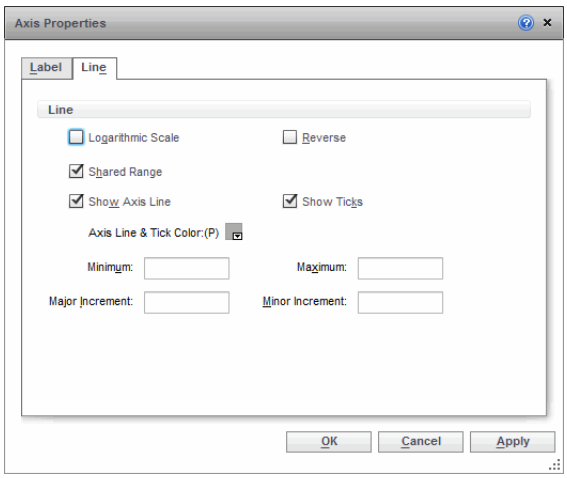
The **Label** tab of the ‘Axis Properties’ dialog box offers the following options:

Show Axis Labels	Display text labels on the axis.
Rotation	The angle of the label text.



The **Line** tab of the 'Axis Properties' dialog box offers the following options:.

Logarithmic Scale	Set logarithmic scaling for the axis. (The logarithm base is selected automatically.)
Reverse	Orient the axis with maximum at bottom, minimum at top.
Shared Range	For a facet-type chart (i.e., a chart with both a measure and dimension on the same axis or with multiple dimensions on the same axis), this specifies that sub-charts in the same column share the same X-axis scaling, and sub-charts in the same row share the same Y-axis scaling. Otherwise, sub-charts are individually scaled.
Ignore Null	For Date/Time and Categorical axes, 'Ignore Null' suppresses the display of null dimension values. When 'Ignore Null' is disabled, null values are represented without labels.
Truncate Long Labels	When enabled, truncates labels that exceed 1/3 of the total chart width or height (using ellipses to indicate omitted text). When disabled, truncates labels only if they exceed 2/3 of the total chart width or height.
Show Axis Line	Display the axis line, in the specified color.
Show Ticks	Display tick marks on the axis line.
Axis Line & Tick Color	Color of the axis line and tick markers.
Minimum	Lowest value shown on axis
Maximum	Highest value shown on axis
Minor Increment	The interval between axis tick marks
Major Increment	The interval between axis labels. For a date/time field which is represented as a time-series (e.g., when 'As time series' is selected in the 'Edit Dimension' panel), the increment is specified in terms of the prevailing time unit (weeks, months, etc.)



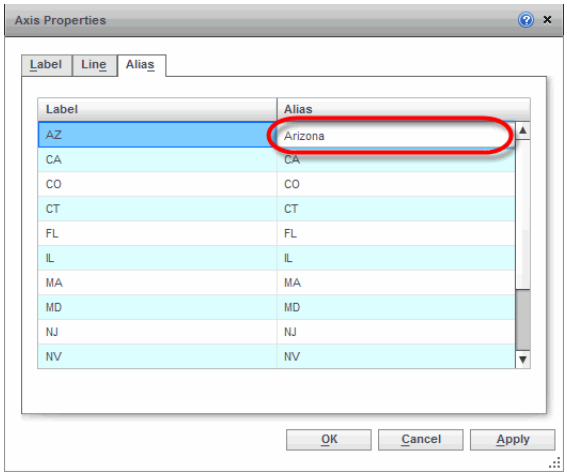
The **Alias** tab is available for dimension (categorical) axes.

The **Alias** tab of the 'Axis Properties' dialog box allows you to change the labels displayed on the axis. To modify the labels of a categorical axis, follow the steps below.

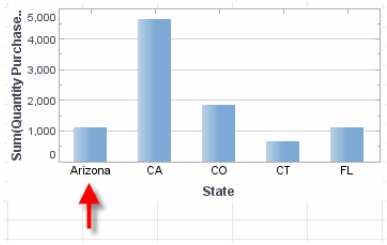
1. Right-click the axis, and select 'Axis Properties'. This opens the 'Axis Properties' dialog box.

2. Click the **Alias** tab. This displays a table containing the mapping between original axis labels and new axis labels.

3. Double-click the cells in the 'Alias' column that correspond to the axis labels that you wish to rename, and enter the desired replacement text.



4. Click 'OK' to close the dialog box. This updates the axis labels with the specified replacement text.



See Also

Editing a Chart Legend, for information about modifying legend labels.

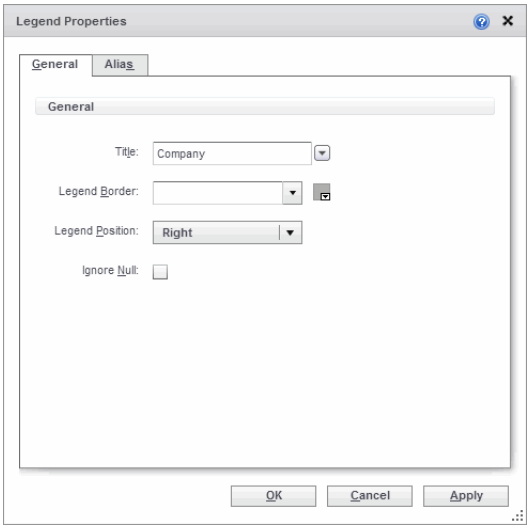
Editing a Chart Legend

To edit a legend, right-click the legend, and select 'Legend Properties' from the context menu. This opens the 'Legend Properties' dialog box.

Note: To hide a legend, right-click the legend and choose ‘Hide Legend’ from the context menu. To show all legends, right-click the chart and select ‘Show Legend’ from the context menu.

The **General** tab of the ‘Legend Properties’ dialog box offers the following options:

Title	The text to display in the legend title bar.
Legend Border	The line style and color of the legend boundary.
Legend Position	Sets the position of the legend on the plot. Legend position can also be changed by dragging with the mouse.
Ignore Null	Ignore null values when generating the legend.



The **Content** tab of the ‘Legend Properties’ dialog box offers the following options:

Font	The font and color of the legend content text.
Format	The format of the legend contents. For information about the available format specifications, see the *Format Tab* section.

The **Scale** tab is available for measure (numerical) legends.

The **Scale** tab of the ‘Legend Properties’ dialog box offers the following two options:

Logarithmic Scale	Set logarithmic scaling for the legend scale. (The logarithm base is selected automatically.)
Reverse	Orient the scale with maximum at left, minimum at right.

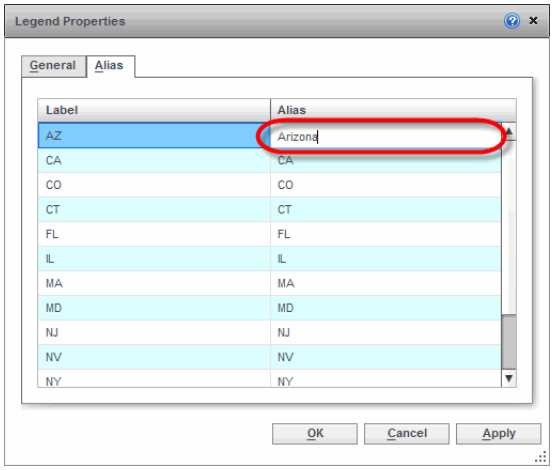
The **Alias** tab is available for dimension (categorical) legends.

The **Alias** tab of the ‘Legend Properties’ dialog box allows you to change the labels displayed in the legend. To modify the labels of a legend, follow the steps below.

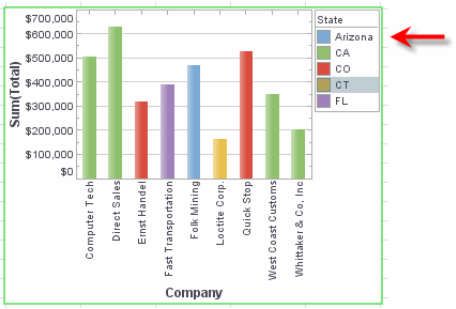
1. Right-click the legend, and select ‘Legend Properties’. This opens the ‘Legend Properties’ dialog box.

2. Click the **Alias** tab. This displays a table containing the mapping between original legend labels and new legend labels.

3. Double-click the cells in the 'Alias' column that correspond to the legend labels that you wish to rename, and enter the desired replacement text.



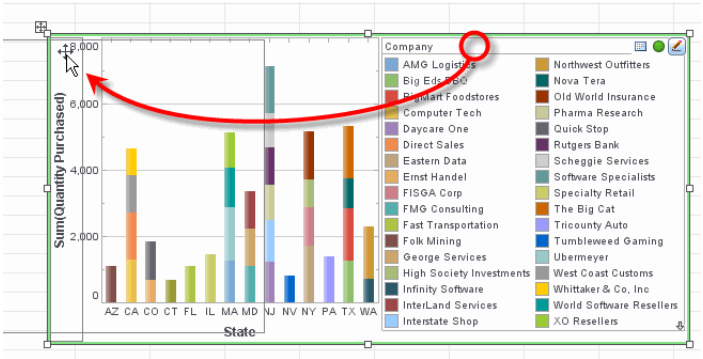
4. Click 'OK' to close the dialog box. This updates the legend labels with the specified replacement text.



See Also
Adding a Sub-Series, for information about adding a sub-series and legend.

Positioning a Chart Legend

To change the position of a chart legend, click the legend title and drag it to the desired location. You can place the legend at the left, right, top, or bottom.



You can also change the legend position from the **General** tab of the 'Legend Properties' dialog box. See *Editing a Chart Legend* for more information.

See Also

Adding a Sub-Series, for information about adding a sub-series and legend.

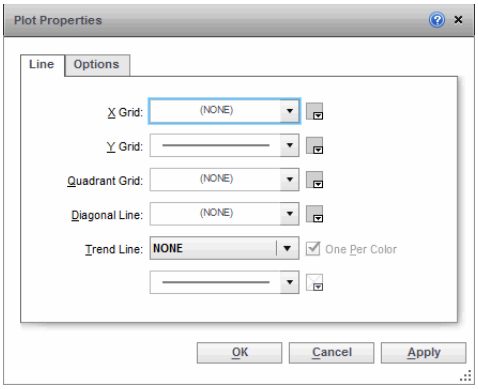
Editing Plot Properties

To edit plot properties, right-click body of the chart, and select 'Plot Property' from the context menu. This opens the 'Plot Properties' dialog box, which contains two tabs: **Line** and **Options**.

The following properties are found under the **Line** tab of the 'Plot Properties' dialog box. (The properties shown depend on the chart style.)

X Grid	The line style and color of the X-axis grid.
Y Grid	The line style and color of the Y-axis grid.
Quadrant Grid	The style and color for a set of perpendicular line segments which divide the plot area into quadrants.
Diagonal Line	The style and color for a line segment which originates at the lower-left corner of the plot area and terminates at the upper-right corner of the plot area.
Show Table Grid	For a facet-type chart (i.e., a chart with both a measure and dimension on the same axis or with multiple dimensions on the same axis), displays a solid line around the entire plot area and displays a solid axis line to separate the dimension and measure axis labels. Use the adjacent color chip to select the color for the line surrounding the plot area. To set the color of the axis separator line, right click on the line and select 'Axis Properties' from the context menu. See *Editing a Chart Axis* for details about setting the axis color.

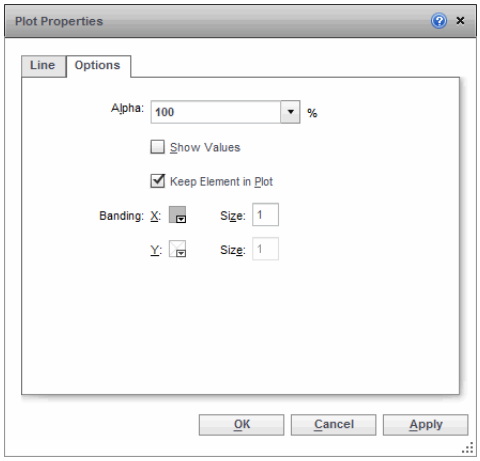
Trend Line	Specifies whether an interpolated trend line is displayed, and the method by which the trend line should be fit to the data (linear, quadratic, etc.). The menu and color chip below the 'Trend Line' menu specify the trend line style and color. If there is a dimension associated with the chart's 'Color' binding, select the 'One Per Color' option to create an independent trend line for each color group. (The trend line colors are matched to the corresponding data colors unless you explicitly specify a trend line color. In that case, all trend lines share the same color.)



The following properties are found under the **Options** tab of the 'Plot Properties' dialog box. (The properties shown depend on the chart style.)

Show Reference Line	Displays horizontal and vertical reference lines when the user moves the mouse over a data point.
Explode Pie	Displays a pie chart with small gaps between the slices.
Alpha	Sets transparency for the chart elements. If disabled, the default transparency for the particular chart type is used.
Show Values	Displays the aggregate measure values on the chart. If the chart also possesses a 'Text' binding, then the values displayed by 'Show Values' and those displayed by the 'Text' binding appear on different lines. Use a 'Text' format to individually format the values. For example, if the top value represents a date and the bottom value represents a number, you can apply the following text format: "Date: {0,date,yyyy}\nAmount: {1,number,$#}". (Note: By default, measure values are also displayed as a tooltip when the user hovers the mouse over the chart. Tooltips are visible when the Viewsheet is previewed or viewed in the Portal.)
Stack Value	Displays a single aggregate value for each primary (X-axis) grouping, rather than for each individual subseries grouping.
Show Points	For a Line-type chart, adds markers at the locations of the individual data points.
Show Lines	Displays line segments connecting the points on the chart. To display a fitted line, use the 'Trend Line' option instead.

Fill Time-Series Gaps	When the 'As time series' option is set for a dimension (see *Editing a Dimension*), the 'Fill Time-Series Gaps' option specifies how missing data should be represented on the chart. When the option is disabled, plotted lines simply ignore the missing data and connect adjacent points. When the option is enabled, you can choose to represent gaps with either 'Null', which leaves an empty space at the location, or with 'Zero', which inserts a numerical value of 0 at the location.
Keep Element in Plot	Automatically extend the chart boundaries to display chart elements (including target lines) that exceed the specified axis maximum.
Always Show Color in Map	When a column is bound to the 'Color' field of the **Visual** panel in the Chart Editor, this setting ensures that the color is always used to fill the map regions. Otherwise, if another column is simultaneously bound to the 'Shape', 'Size', or 'Text' fields of the **Visual** panel, the color generated for the 'Color' binding will be applied to the plotted points rather than to the map regions.
Banding	Adds alternating colored bands to a specified categorical or timescale axis. Select the desired band color from the color picker. The band 'Size' setting is an integer indicating the number of category groups (for a categorical axis) or the number of date groups (for a timescale axis) to be spanned by each band.



See Also

Editing Chart Formats, for information about editing axis data.
Setting the Chart Style, for information on switching chart styles.
Changing the Chart View, for information about editing axis data.
Color Tab, for information on setting component fill and transparency.
Chart Properties, for general properties of the Chart component.

3.4.5 Changing the Chart View

The following sections explain how to zoom in on a particular region of the chart, and how to explore the data underlying the chart.

Selecting Chart Data

You can select contiguous or non-contiguous data in a chart. There are several ways to select data:

- Drag the mouse across the data you wish to select. Ctrl-drag to add additional regions to the selection.
- Ctrl-click the data you wish to select.
- Ctrl-click the X-axis labels to select the corresponding data.
- Ctrl-click the legend labels to select the corresponding data (subseries).

See Also

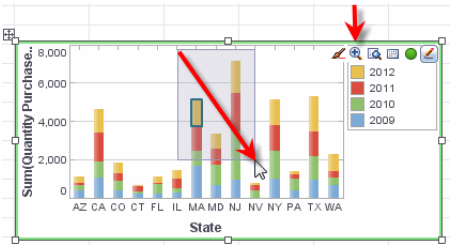
Zooming a Chart, for information on zooming in on selected chart data.
Showing Detail Data, for information on viewing records of selected data.
Brushing a Chart, for information about highlighting selected data across charts.

Zooming a Chart

To zoom in on a particular region of the chart, follow the steps below:

1. Select the group(s) on the chart that you wish to zoom. (See *Selecting Chart Data*.)

2. Click the 'Zoom' button in the toolbar. This zooms the chart to display only the data selected.



3. Repeat the above steps to zoom in further.

4. To return to the original chart view, press the 'Clear Zoom' button.



Example:
Zooming a Chart

The figures below shows a variety of different selections, and the zoomed results.

Figure 4. Zooming on Subseries Categories (Legend Labels)

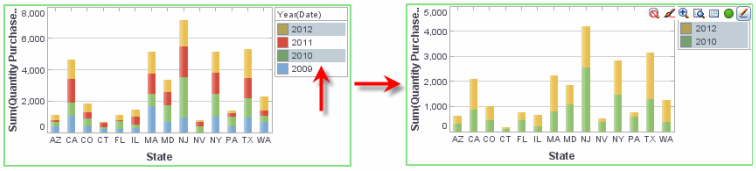
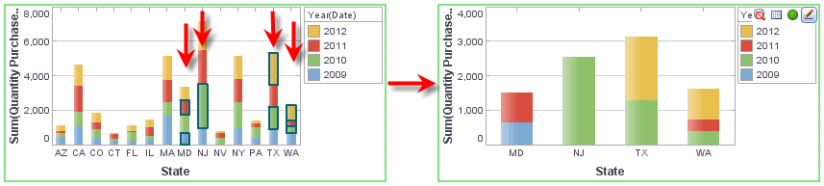


Figure 5. Zooming on X-axis Categories (Labels)



Figure 6. Zooming on Arbitrary Categories



¢

See Also

Excluding Data From a Chart, to create a zoomed view by excluding groups.

Selecting Chart Data, for information on methods of selecting chart data.

Filter Conditions, and *Filtering a Chart* in *Interacting and Collaborating*, for information on filtering chart data

Adding Highlights to a Chart, for information on conditional formatting.

Brushing a Chart, for information on highlighting data across multiple views.

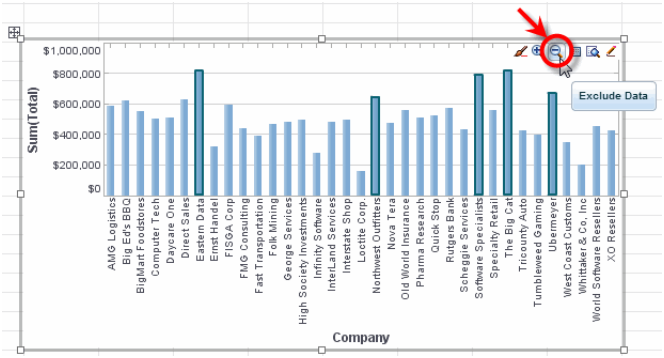
Excluding Data From a Chart

The Zoom feature (*Zooming a Chart*) allows you focus in on a selected set of groups by specifying the groups to *include* in the zoomed view. In some cases, however, you might find it easier to create the desired view by specifying particular groups to *exclude* from the zoomed view.

To exclude particular groups from a chart display, follow the steps below:

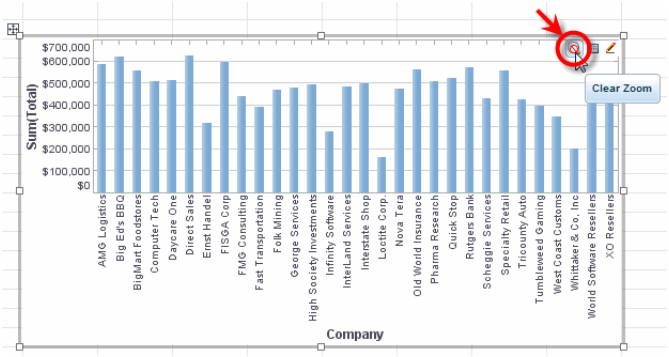
1. Select the group(s) in the chart that you wish to exclude. (See *Selecting Chart Data*.)

2. Press the 'Exclude Data' button in the toolbar. This removes the selected data from the chart.



3. Repeat the above steps to exclude further data.

4. To return to the original chart view, press the 'Clear Zoom' button.



See Also

Zooming a Chart, to create a zoomed view by including groups.
Selecting Chart Data, for information on methods of selecting chart data.
Filter Conditions, and *Filtering a Chart* in *Interacting and Collaborating*, for information on filtering chart data
Adding Highlights to a Chart, for information on conditional formatting.
Brushing a Chart, for information on highlighting data across multiple views.

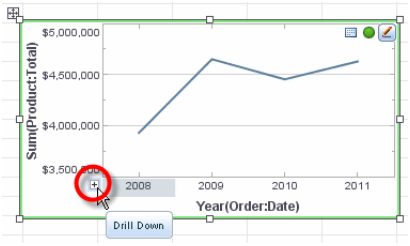
Drilling Down into a Chart

When a chart axis displays a date field or a cube dimension, the chart automatically provides the user with the ability to drill down to finer levels of resolution of the date or dimension.

To drill down into a date or cube dimension on a chart axis, follow these steps:

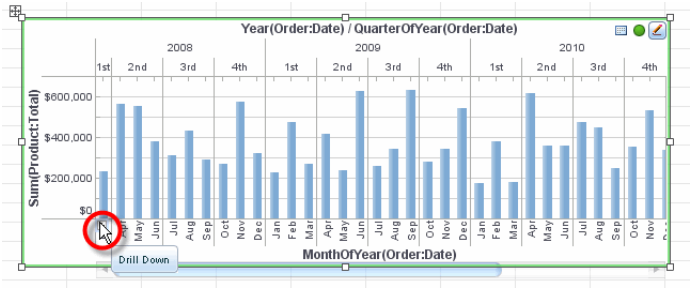
The '+' symbol is only visible when you hover over the axis.

1. Hover the mouse over the chart axis that you want to display at greater resolution. For axes that represent date or cube dimension data, this will display a '+' symbol at the left side of the axis.

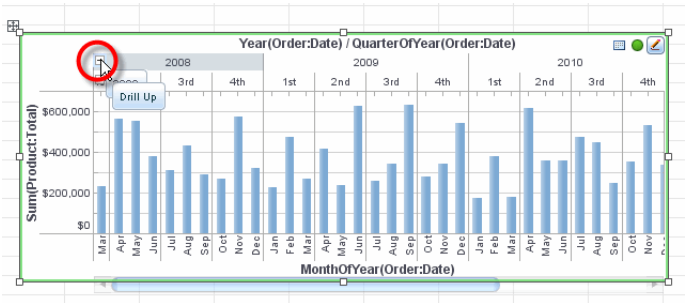


2. Click the '+' symbol to drill down to the next level of the date or dimension.

3. Repeat the above steps to drill down to deeper levels of the hierarchy.



4. To drill up to a higher level of the date or dimension hierarchy, hover the mouse over the chart axis that you want to display at reduced resolution. This displays a '–' symbol at the left side of the axis.



5. Click the '–' symbol to drill up to the higher level of the date or dimension hierarchy.

To disable the drilling feature for a particular chart, deselect the 'Enable Drilling' option in the **Advanced** tab of the 'Chart Properties' dialog box. To specify a custom drill hierarchy, including custom date levels, see *Chart Properties: Hierarchy Tab*.

See Also

Chart Properties: Hierarchy Tab, for information on custom drill hierarchies.

Drilling Down into a Crosstab, to perform the drill operation on a crosstab dimension.

Walkthrough: Interactive Chart Drilldown, in *Advanced Dashboard Scripting*, to implement a combined drill-and-filter operation.

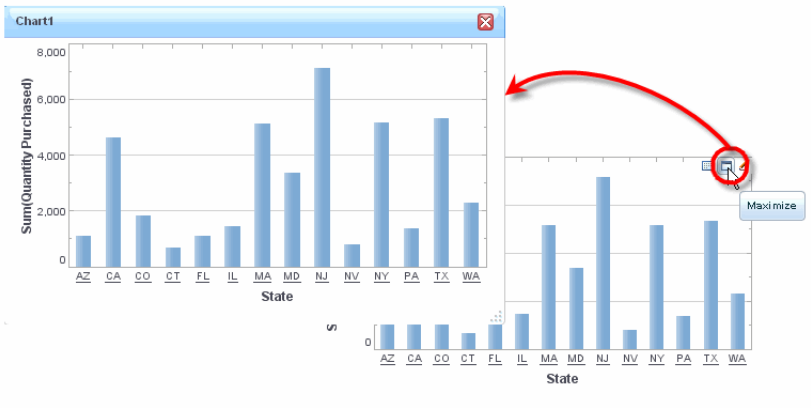
Filter Conditions, and *Filtering a Chart* in *Interacting and Collaborating*, for information on filtering chart data

Adding Highlights to a Chart, for information on conditional formatting.

Brushing a Chart, for information on highlighting data across multiple views.

Maximizing a Chart

The 'Max Mode View' button at the top-right corner of the chart allows you to detach the chart from the Viewsheet grid. This causes the chart to "float" above the grid as a separate window that you can freely move and resize.



To reattach a floating chart to the Viewsheet grid, click the close button in the top-right corner of the floating window.

The 'Max Mode View' feature is available to end-users as well, which allows you to create smaller charts ("thumbnails") that the user can enlarge as needed.

See Also

Positioning a Component, for information on scaling and moving components.

Showing Chart Data

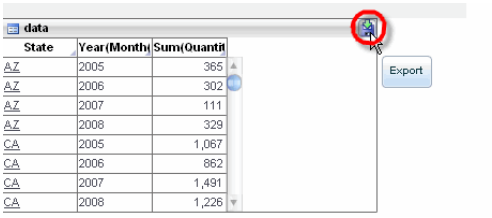
Tip: To copy the contents of a chart label, click on the label.

To display a chart's (summarized) data in tabular form, press the 'Show Data' button at the top-right of the chart. To close the data panel, press the red 'X' in the top-right corner.



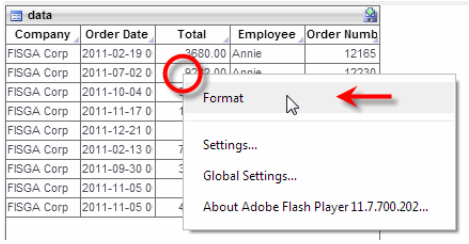
Exporting Chart Data

To export the chart data, click the 'Export' button.

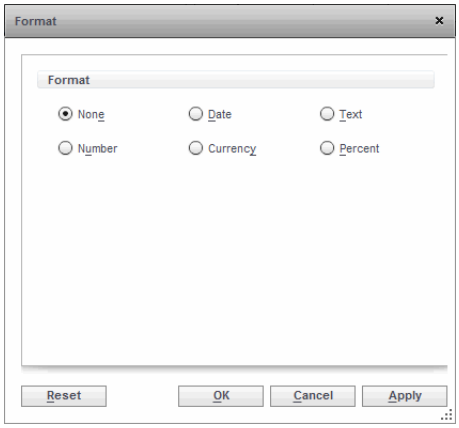


Formatting Chart Data

To format values in the data panel, right-click on the column you wish to format, and select 'Format' from the context menu.



This opens the 'Format' dialog box, which allows you to set the format for the column. See *Format Tab* for information about the available options.



Formatting that you specify is saved with the Viewsheet, and is automatically applied to the column whenever the 'Show Data' option is used, both in Visual Composer and the User Portal.

See Also

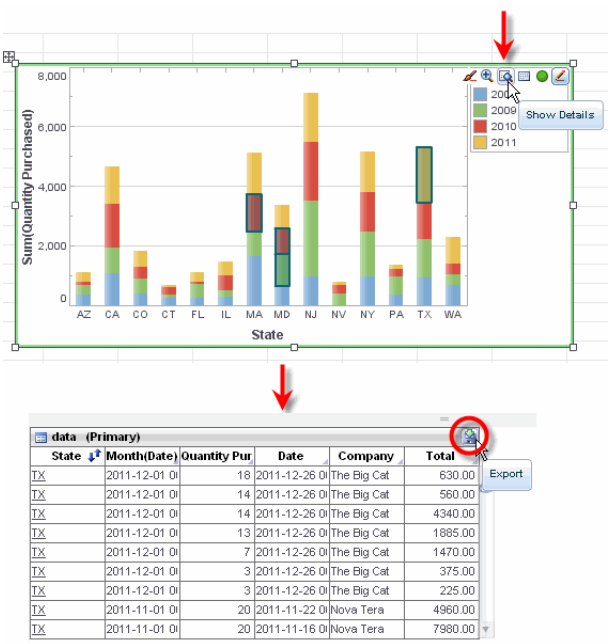
Showing Detail Data, for information on viewing underlying detail records.

Showing Detail Data

To display the detail records corresponding to the chart's (summarized) data, follow the steps below:

1. Select the data to display by clicking the appropriate element in the chart.

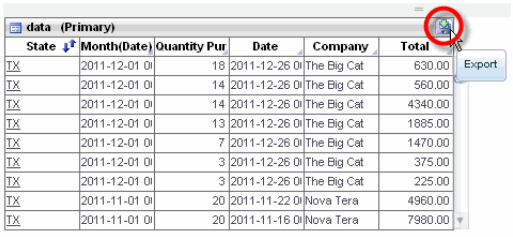
2. Click the 'Show Details' button at the top-right of the chart.



To close the data panel, click the red ‘X’ in the top-right corner.

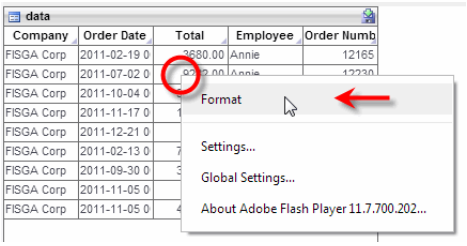
Exporting Detail Data

To export the data, click the ‘Export’ button.

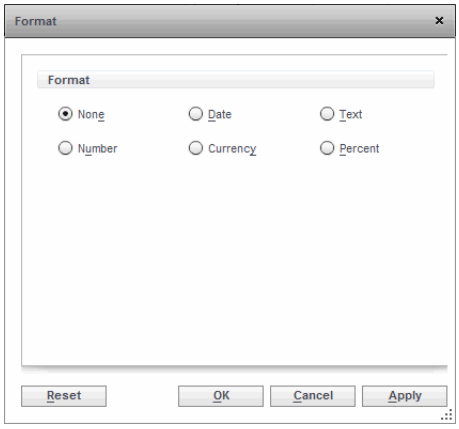


Formatting Detail Data

To format values in the data panel, right-click on the column you wish to format, and select ‘Format’ from the context menu.



This opens the ‘Format’ dialog box, which allows you to set the format for the column. See *Format Tab* for information about the available options.



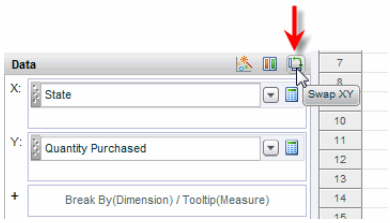
Formatting that you specify is saved with the Viewsheet, and is automatically applied to the column whenever the 'Show Details' option is used, both in Visual Composer and the User Portal.

See Also

Showing Chart Data, for information on viewing summarized chart data.
Selecting Chart Data, for information on methods of selecting chart data.

Rotating a Chart

To rotate a chart (switch the X- and Y-axes), click the 'Swap XY' button in the **Data** panel title bar.



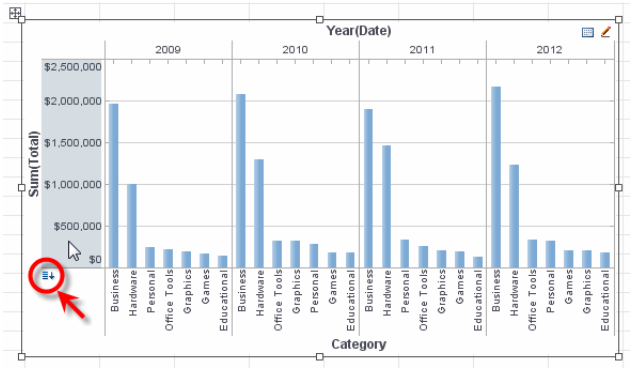
This will position the X-axis vertically, and position the Y-axis horizontally.



Sorting a Chart Dimension by Value

To sort chart groups based on a measure, hover the mouse over the desired measure axis. This displays the 'Sort' button next to the axis. Click the button once to sort the groups in ascending order by measure value, click a

second time to sort in descending order by measure value, and click a third time to restore the original order.



When multiple dimensions are represented on the chart (as in the image above), the ‘Sort’ button operates on the innermost dimension.

See Also
Editing a Dimension, to select from additional sorting options.

Clearing Chart Data

To clear all data binding information from a chart, right-click the chart, and select ‘Clear All Data’ from the context menu. This restores the chart to the its initial blank state.

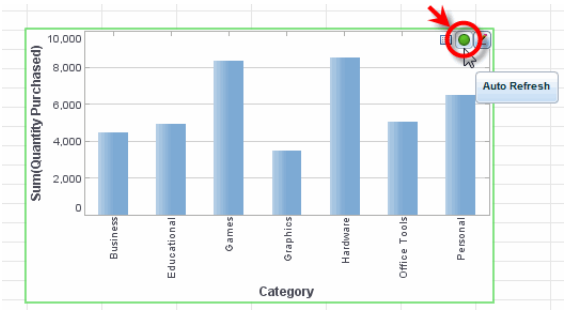
See Also
Editing a Chart Axis, for information on hiding axes.

Manually Refreshing a Chart

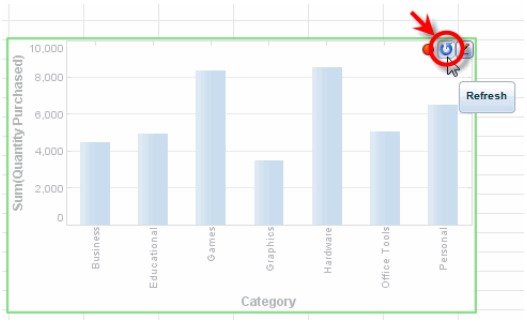
By default, a chart is refreshed automatically whenever you make a change to the chart binding. In cases where the automatic refresh creates an undesirable delay, you can disable the automatic refresh feature.

To disable automatic refresh for a chart (and perform the refresh operation manually), follow these steps:

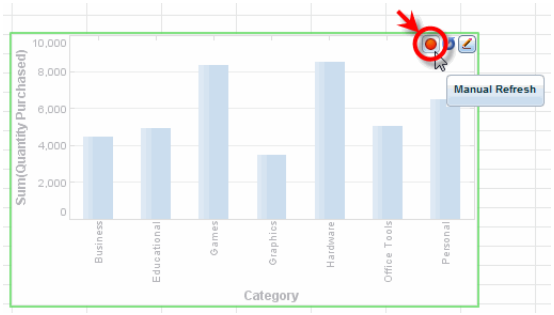
1. Press the ‘Edit’ button on the chart to open the Chart Editor.
2. Press the ‘Auto Refresh’ button in the chart toolbar to disable the auto-refresh feature.



3. To manually refresh the chart, press the ‘Refresh’ button.



To re-enable the auto-refresh feature, press the ‘Manual Refresh’ button on the chart.



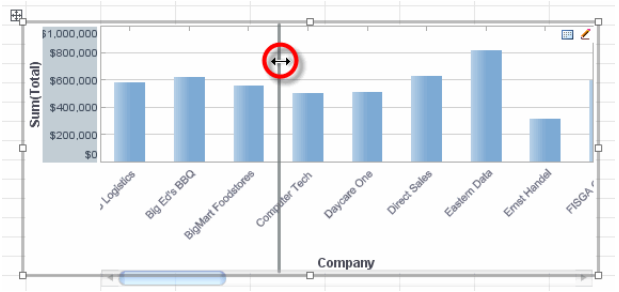
See Also
Editing a Chart Axis, for information on hiding axes.

Changing Chart Element Spacing

To change the vertical or horizontal spacing of elements on a chart, follow the steps below:

1. Hover the mouse into an empty region between the chart elements. The “arrow” pointer will change into a “move” pointer.

2. Click and drag to change the element spacing. Drag left or right to change horizontal spacing. Drag up or down to change vertical spacing.



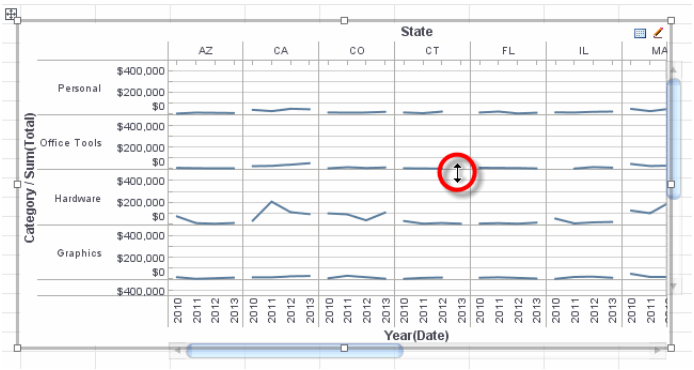
The chart will automatically enable scroll bars as required to accommodate the new spacing.

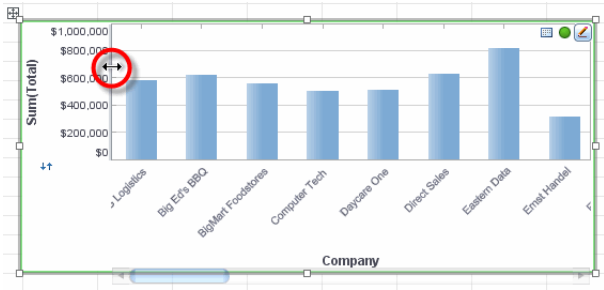
Changing Chart Axis Spacing

To change the position of a chart axis or control spacing among axes on a compound chart, follow the steps below:

1. Hover the mouse over the horizontal or vertical axis you want to move. The “arrow” pointer will change into a “move” pointer.

2. Click and drag to move the axis. Drag left or right to change horizontal spacing. Drag up or down to change vertical spacing.





The chart will automatically enable scroll bars as required to accommodate the new spacing.

Grouping Chart Labels into Named Groups

You can agglomerate data from different chart groups (axis groups, legend groups) into larger categories called *named groups*. For example, you can combine X-axis labels designating individual states MA, CT, NY, NJ, etc., into larger groups such as ‘New England’ and ‘Mid-Atlantic’.

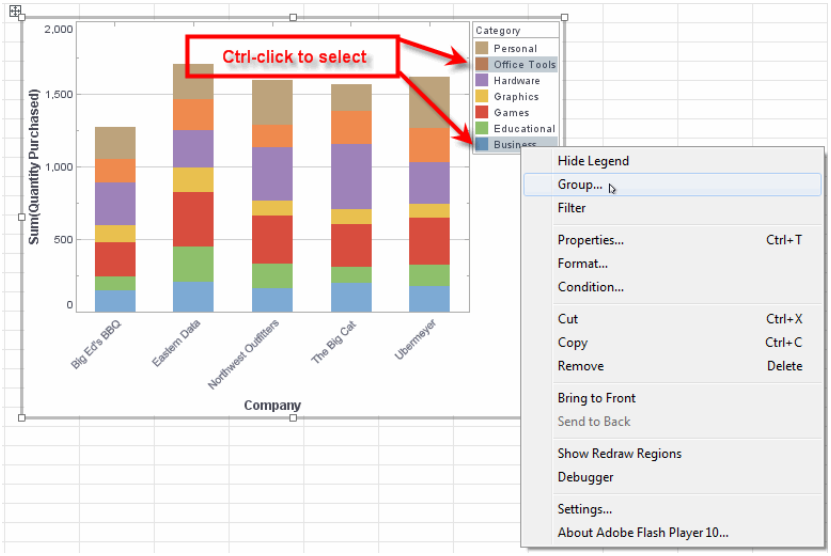
```
Named groups cannot be defined for date fields.
```

To create named groups for chart axis labels or legend labels, follow the steps below:

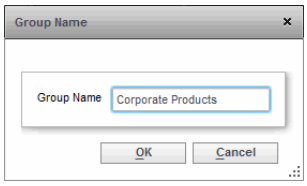
1. Ctrl-click to select the individual axis labels or legend labels that you want to combine into a single group.

You can group existing named groups to make larger groups.

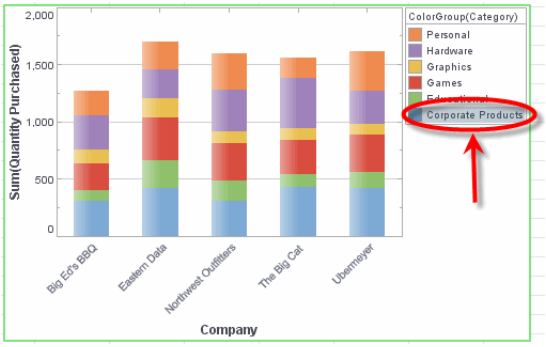
2. Right-click on one of the selected labels, and choose ‘Group’ from the context menu. This opens the ‘Group Name’ dialog box.



3. Enter a name for the new group in the ‘Group Name’ dialog box, and click ‘OK’.



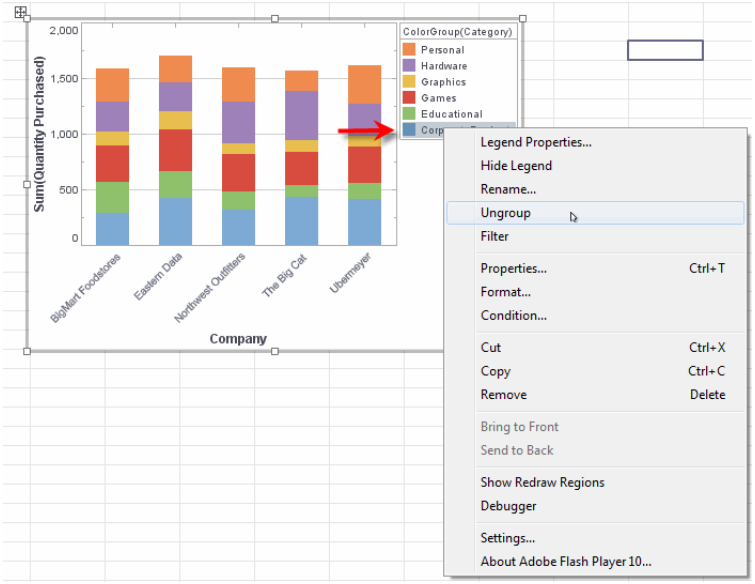
This agglomerates the data corresponding to the original labels and replaces the original labels with the named grouping label.



4. Repeat the above steps to add additional named groups.

To rename an existing group, right-click the group label, and select 'Rename' from the context menu. Alternatively, select 'Axis Properties' or 'Legend Properties' from the context menu, and change the label's alias on the **Alias** tab of the 'Properties' dialog box. (See *Editing a Chart Axis* or *Editing a Chart Legend* for more details.)

To decompose an existing named group, right-click on the group label and select 'Ungroup' from the context menu.



Example: Creating Named Groups

In the following example, you will create a chart that displays total sales by state. You will then group the states into one of six standard regions: New England, Mid-Atlantic, Midwest, South, Mountain, and Pacific. Follow the steps below:

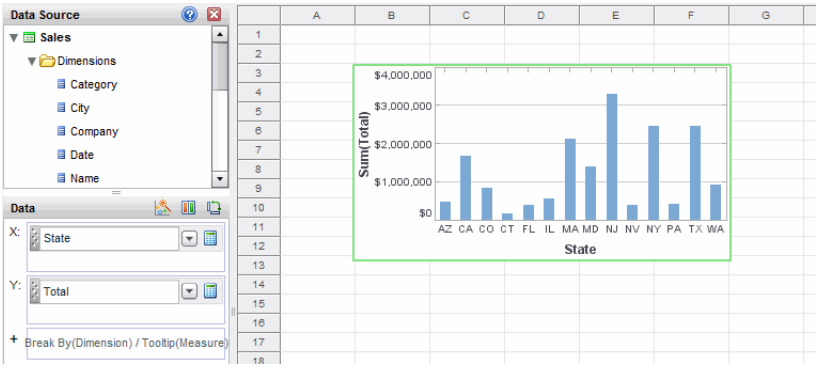
1. Create a new Viewsheet based on the 'Sales' > 'Sales Explore' Worksheet.

2. Drag a Chart element from the **Component** panel into the Viewsheet grid.

3. Click the 'Edit' button on the Chart to open the Chart Editor.

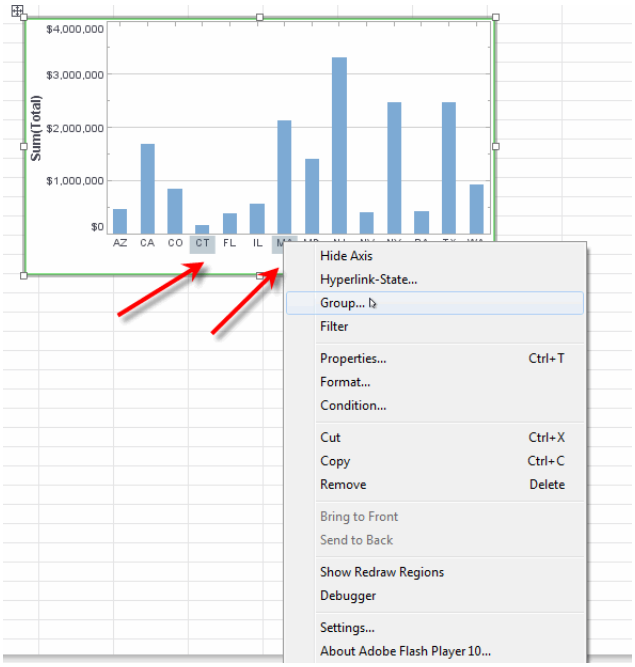
4. In the **Data Source** panel, expand the 'Sales' Data Block. From the 'Dimensions' node, drag the 'State' field to the 'X' region of the **Data** panel.

5. From the 'Dimensions' node, drag the 'Total' field to the 'Y' region of the **Data** panel.



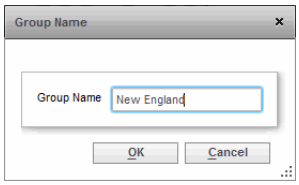
6. Click the 'CT' label on the Chart's X-axis to select it. Ctrl-click on the 'MA' label to select it also.

7. Right-click on either of the selected labels, and choose 'Group' from the context menu.

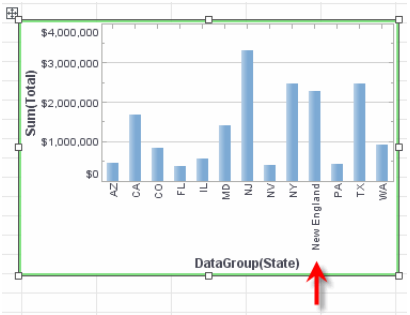


This opens the ‘Group Name’ dialog box.

8. In the ‘Group Name’ dialog box, enter the label for the new composite group: “New England”.



This adds ‘New England’ as a new X-axis label, and agglomerates the data from ‘MA’ and ‘CT’ into this new composite group.



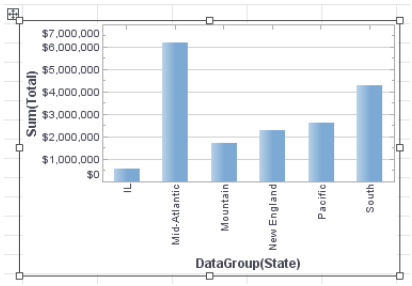
9. Repeat the above steps to create the following additional named groups:

   a. “Mid-Atlantic,” including states ‘NY’, ‘PA’, ‘NJ’.

   b. “South,” including states ‘MD’, ‘TX’, ‘FL’.

   c. “Mountain,” including states ‘AZ’, ‘CO’, ‘NV’.

   d. “Pacific,” including states ‘CA’, ‘WA’.



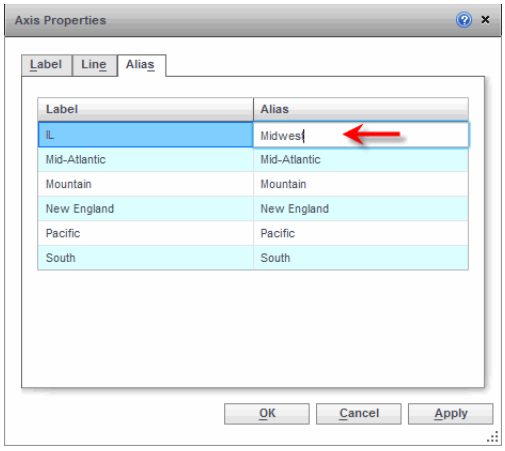
All states are now grouped into larger regions, except for ‘IL’, which belongs to the Midwest region. You cannot create a named group for ‘Midwest’ on the basis of a single label. Instead, if you

need a group labeled ‘Midwest’, you can simply rename the ‘IL’ label as described in the next step.

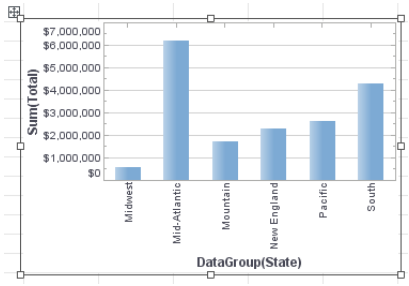
10. (Optional) Rename the ‘IL’ label to “Midwest”. Follow the steps below:

    a. Right-click the ‘IL’ label on the X-axis, and select ‘Axis Properties’ from the context menu. This opens the ‘Axis Properties’ dialog box.

    b. In the ‘Axis Properties’ dialog box, select the **Alias** tab. Double-click in the ‘Alias’ column next to the ‘IL’ label, and enter the text “Midwest”.



    c. Click ‘OK’ to close the dialog box. The Chart now displays all states grouped by region.



¢

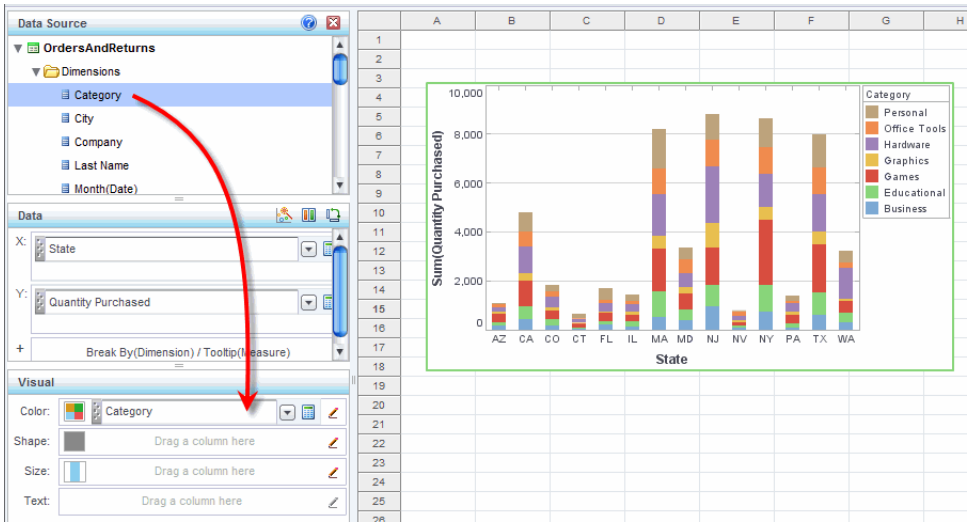
See Also

Grouping Crosstab Headers into Named Groups, to create groupings in a crosstab.

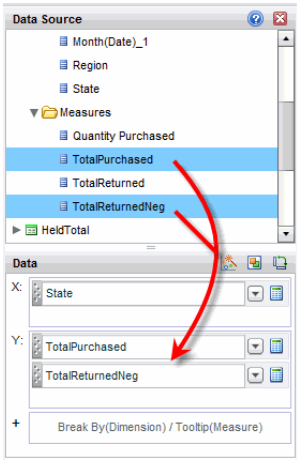
Creating a Chart Legend

To create a chart legend, do one of the following:

- Drag a measure or dimension to one of the regions in the **Visual** panel of the Chart Editor (‘Color’, ‘Shape’, or ‘Size’). See *Working with Multiple Measures and Subseries* for more information.



- Drag two or more measures to the ‘X’ or ‘Y’ regions in the **Data** panel of the Chart Editor.



For information about modifying the legend, see *Editing a Chart Legend*.

See Also

Editing a Chart Legend, for information about modifying a chart legend.
Positioning a Chart Legend, for information about moving a chart legend.

3.4.6 Brushing a Chart

A Viewsheet often contains multiple charts representing the same data in different ways. In this case, it is often useful to highlight corresponding data in different plots. You can do this easily with a technique called “brushing.”

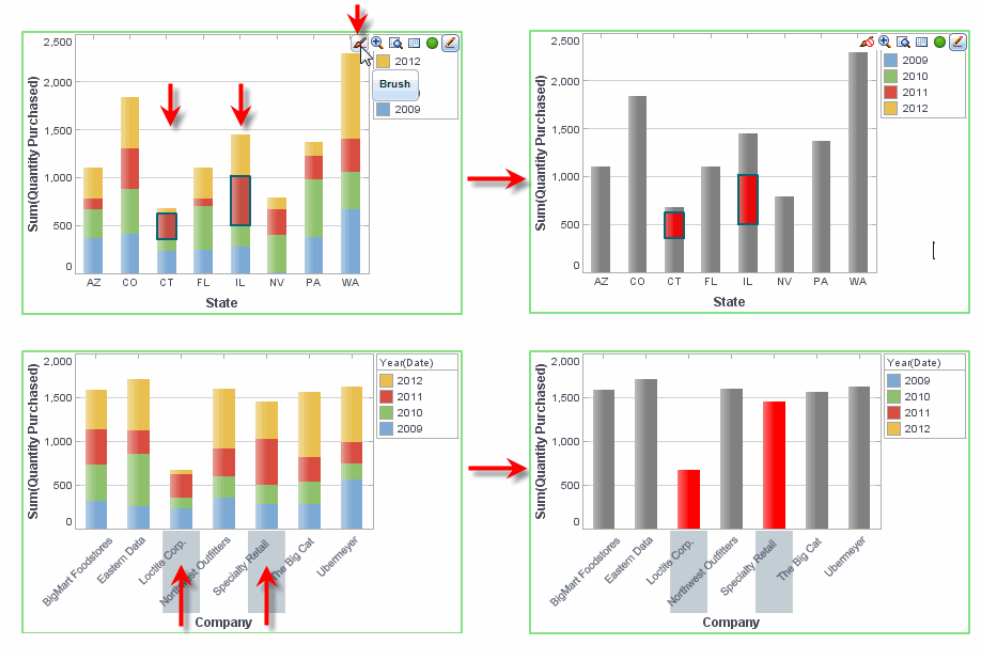
To brush a single subgroup, double-click the desired entity (axis label, legend label, or subgroup). To brush a set of subgroups, follow the steps below:

1. Select the desired subgroups, adjacent or nonadjacent. See *Selecting Chart Data*.

2. Click the 'Brush' button in the chart toolbar.

The corresponding data in all charts is highlighted in red, allowing you to immediately identify key data relationships. Data represented in tables and output elements (e.g., gauges) is also filtered by the brushing operation.

Figure 7. Brushing data: Companies in CT and IL for 2007.



Example: Brushing

Brushing is a powerful tool that helps you quickly understand your data. In this example, you will create two charts, and then use brushing on both charts to answer particular business questions.

1. Create a new Viewsheet based on the sample 'Analysis' Worksheet.

2. To create the first chart, follow the steps below:

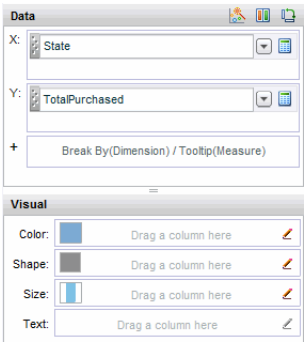
   a. Add a Chart component, and click the 'Edit' button. This opens the Chart Editor.

b. In the **Data Source** panel, expand the 'OrdersAndReturns' data block.

c. From the 'Dimensions' node in the **Data Source** panel, drag the 'State' field to the 'X' field of the **Data** panel.

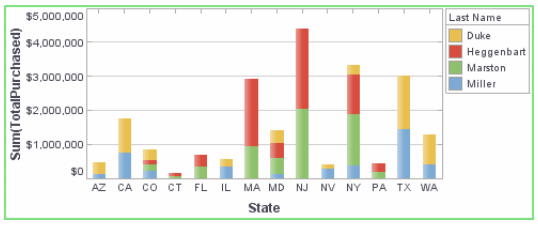
d. From 'Dimensions' node in the **Data Source** panel, drag the 'Last Name' field to the 'Color' field of the **Visual** panel.

e. From the 'Measures' node in the **Data Source** panel, drag the 'TotalPurchased' field to the 'Y' field of the **Data** panel.



This automatically creates a stacked bar chart.

f. Enlarge the chart (if desired) by dragging the handles.



3. To create the second chart, follow the steps below:

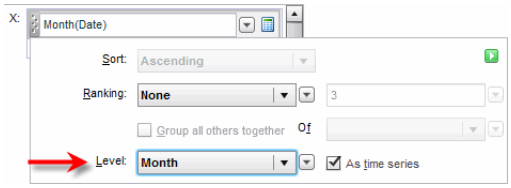
a. Add a Chart component, and click the 'Edit' button. This opens the Chart Editor.

b. In the **Data Source** panel, expand the 'OrdersAndReturns' data block.

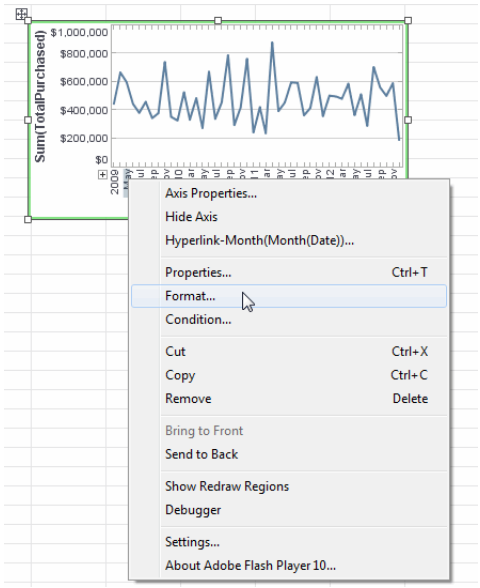
c. From the 'Dimensions' node of the **Data Source** panel, drag the 'Date' field to the 'X' field of the **Data** panel.

d. From the 'Measures' node of the **Data Source** panel, drag the 'TotalPurchased' field to the 'Y' field of the **Data** panel.

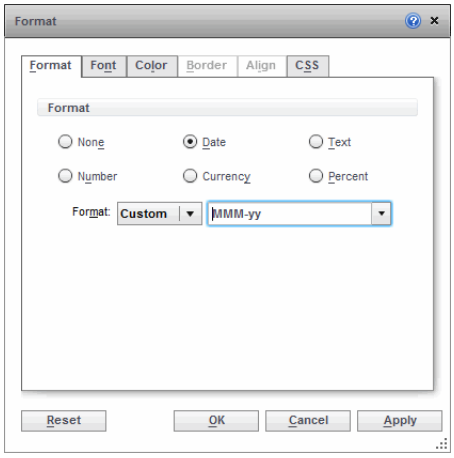
e. Next to the ‘X’ field, click the ‘Edit Dimension’ button. Select ‘Month’ for ‘Level’.



f. On the new chart, right-click an X-axis label and select ‘Format’ from the context menu. This opens the ‘Format’ dialog box, where you will set the axis format to match the grouping level.



g. In the ‘Format’ dialog box, click the **Format** tab, and select the ‘Date’ option. Enter “MMM-yy” as the format, and click ‘OK’.

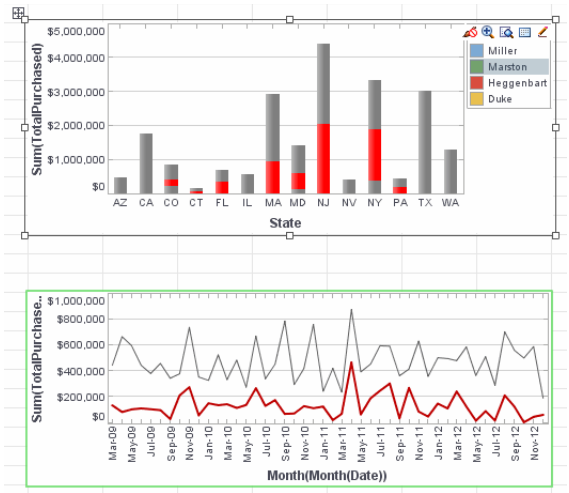


h. On the chart, right-click an X-axis label and select ‘Axis Properties’ from the context menu. This opens the ‘Axis Properties’ dialog box.

i. In the **Label** tab of the ‘Axis Properties’ dialog box, select a slanted label orientation and small font size. This will allow the labels to better display.

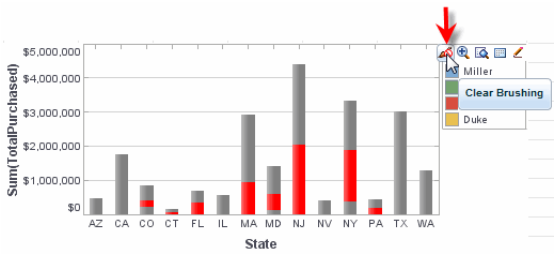
4. Use brushing to answer the following question: “In what month did salesperson Eric Heggenbart have his peak sales?”

   a. Double-click the ‘Heggenbart’ label in the legend to brush Heggenbart’s sales in both charts.



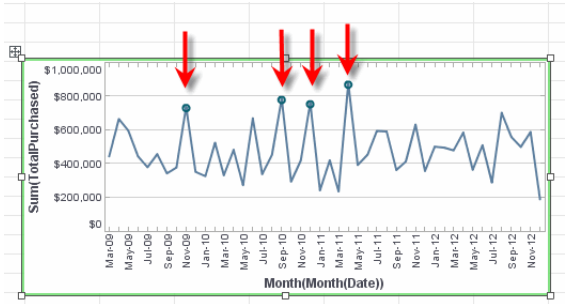
   It is now obvious that Heggenbart’s greatest sales month was December of 2006, when he sold over $400,000.

   b. Press the ‘Clear Brushing’ button to return to the normal view.



5. Use brushing to answer the following question: “Which states had the greatest sales numbers during the best four months?”

   a. In the second chart, Ctrl-click to select the four highest peaks on the plot.



b. Click the 'Brush' button on the chart toolbar to highlight all the data corresponding to these months.

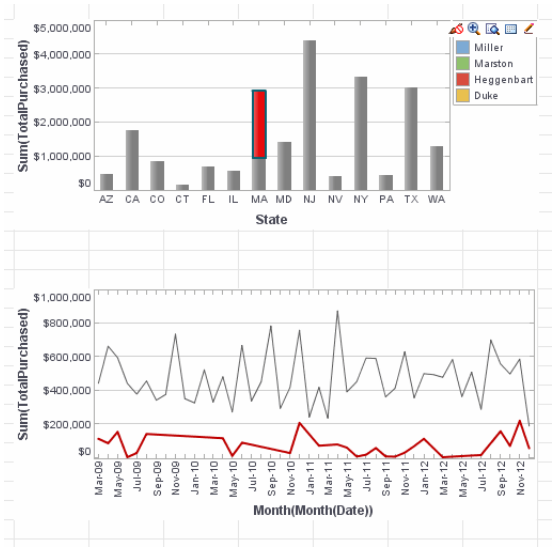


It is clear that the four peak months saw sales chiefly in the states NJ, TX, and MA.

6. The data indicate a steep overall decline in sales for September 2008. Use brushing to answer the following question: "Which salesperson's sales defied this trend, and what state's sales most significantly accounted for this?"

   a. Successively double-click the names in the legend to display the sales by data for each salesperson. Note that only the data for Heggenbart defy the September sales decline.

   b. Successively double-click each of Heggenbart's states, and observe the September sales movement. Note that only the Heggenbart sales in MA and NY show significant growth in this period.



¢

See Also

Zooming a Chart, for information on zooming in on selected data.
Adding Highlights to a Chart, for information on adding conditional formatting.
Flyover View, to use chart selections to filter the data in other components.
Selecting Chart Data, for information on methods of selecting chart data.

3.4.7 Working with Multiple Measures and Subseries

To plot multiple datasets on the same chart, simply drag multiple measures into the 'Y' field of the **Data** panel. To break out a dataset into a subseries, drag the desired subseries dimension into one of the fields in the **Visual** panel ('Color', 'Shape', 'Size', 'Text').

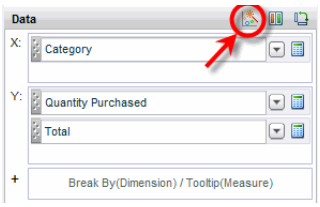
The following sections explain various aspects of working with multiple datasets and subseries.

Setting a Chart Style for an Individual Dataset

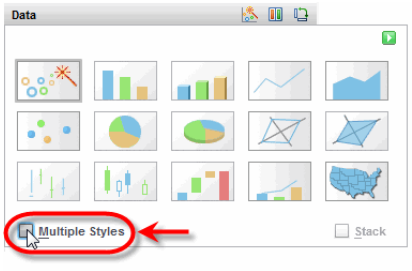
To set the chart style for an individual dataset on a multi-dataset chart, follow the steps below:

Note: The 'Auto' chart type automatically selects an appropriate chart type for the provided data.

1. In the **Data** panel of the Chart Editor, press the 'Select Chart Type' button. This opens the chart styles panel.

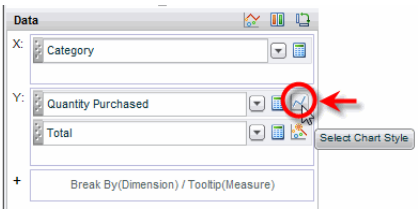


2. In the chart styles panel, enable the ‘Multiple Styles’ option (below the chart style options).

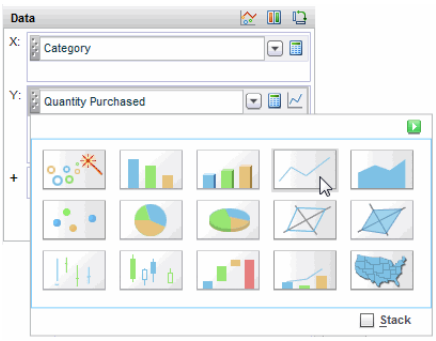


This automatically closes the panel, and enables you to set styles individually for each data set.

3. Press the ‘Select Chart Type’ button next to dataset you wish to change.



This opens the chart styles panel for the specified dataset.




```
Note: Only certain compatible style combinations can be selected
(e.g., Pie style and Line style are incompatible).
```

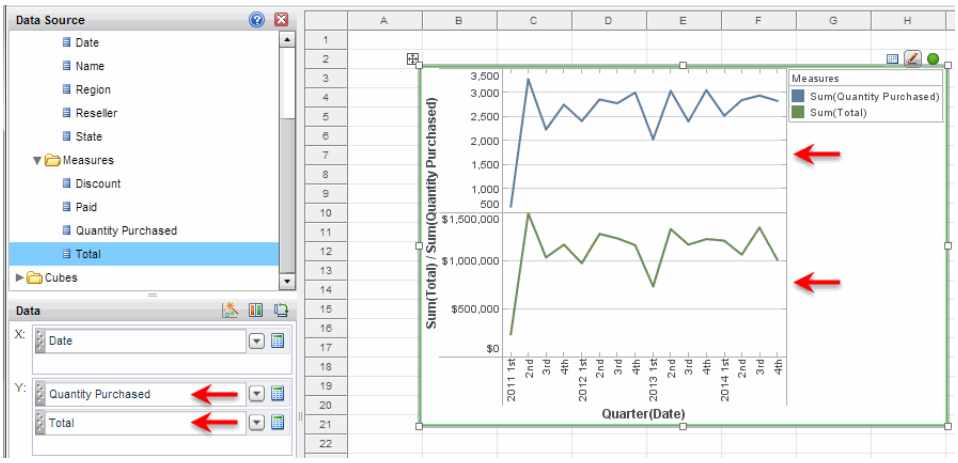
4. Click a chart type to select it, and then click the ‘Apply’ button. (You can also double-click the desired chart type.)

See Also

Setting the Chart Style, for information on setting a single style for all datasets.

Plotting Measures on a Single Axis

By default, when you add two or more measure fields on the Y-axis, the selected datasets are plotted on adjacent sets of axes, as shown below:

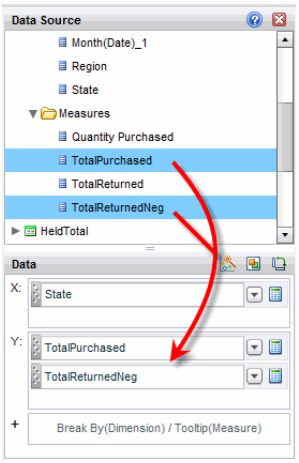


In some cases you may want to display the measures on the same axis. The following example illustrates how to do this.

Walkthrough

This example illustrates how to switch from a multiple axis view to a single-axis view. Follow the steps below:

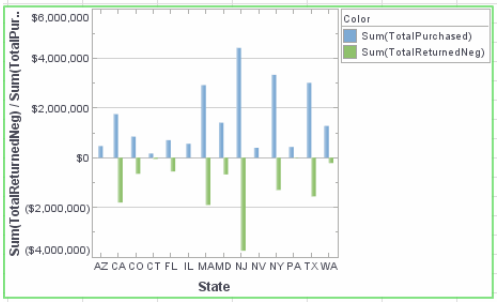
1. Create a new Viewsheet based on the sample ‘Analysis’ Worksheet.
2. Add a chart component, and press the ‘Edit’ button. This opens the Chart Editor.
3. In the **Data Source** panel, expand the ‘OrdersAndReturns’ data block.
4. From the ‘Dimensions’ node, drag the ‘State’ field to the ‘X’ field of the **Data** panel.
5. In the ‘Measures’ node, Ctrl-click to select both the ‘TotalPurchased’ field and the ‘TotalReturnedNeg’.
6. Drag both fields together to the ‘Y’ field of the **Data** panel.



This automatically creates two axes, one for each dataset. Note the scrollbar indicators that appear on the chart. (Enlarge the chart component if you cannot see both axes.)



7. Press the 'Switch to Single Graph' button in the **Data** panel of the Chart Editor. This fits both datasets on the same set of axes.



8. If desired, edit the Y-axis label to fit on the chart.

¢

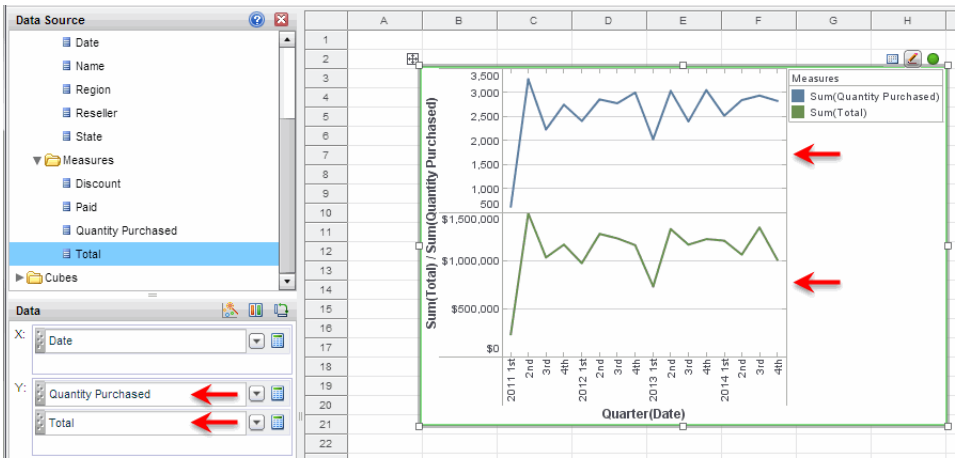
See Also

Plotting Measures on a Secondary Y-Axis, for instructions on how to create a double-axis chart.

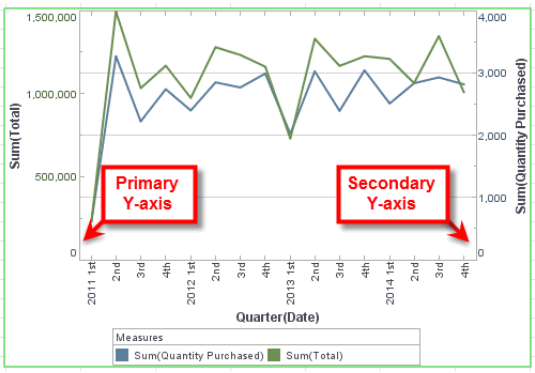
Editing an Axis Title, for information on changing the axis title properties.

Plotting Measures on a Secondary Y-Axis

By default, when you add two or more measure fields on the Y-axis, the selected datasets are plotted on adjacent sets of axes, as shown below:

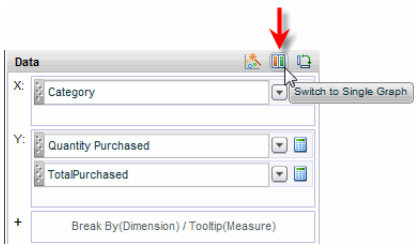


In some cases, you may want to use a double Y-axis to represent these measures, as shown below:



To display a dataset on the right-side (secondary) Y-axis, follow these steps:

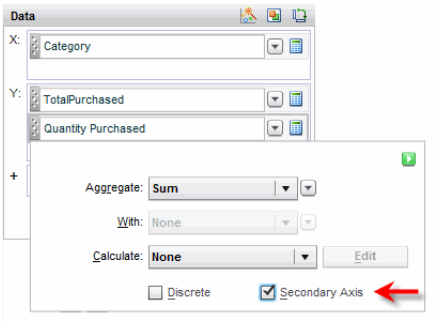
1. Press the 'Switch to Single Graph' button in the **Data** panel of the Chart Editor.



This places all measures on the same Y-axis rather than on adjacent axes.

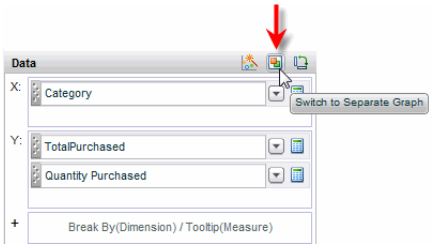
2. Press the ‘Edit Measure’ next to the measure that you wish to display against the secondary Y-axis.

3. Enable the ‘Secondary Axis’ option at the bottom of the panel.



4. Press the ‘Apply’ button (green arrow).

To return to the multiple axis style, press the ‘Switch to Separate Graph’ button.



Controlling Axes in Separate Chart View

In the ‘Separate Graph’ view, you can scroll the chart using vertical and horizontal scrollbars. To see the scrollbars, hover the mouse over the bottom or right of the chart.



To control the spacing of the labels on the chart axes, right-click the chart, and select one of the following options from the context menu:

- Increase Width

- Increase Height
- Decrease Width
- Decrease Height

These options change the axis scale by a small increment. To make a larger change, select the desired option repeatedly.

Adding a Sub-Series

A sub-series represents an additional level of grouping within an individual dataset or measure. Beyond the level of grouping implied by the X-axis labels, the subseries breaks an individual measure down into a secondary level of categories.

A subseries can be created with or without visual formatting.

Adding a Visually-Formatted Subseries

To add a visual-formatted subseries to an existing chart, follow the steps below:

1. Select the chart, and click the ‘Edit’ button to open the Chart Editor.

2. Expand the ‘Dimension’ node in the **Data Source** panel.

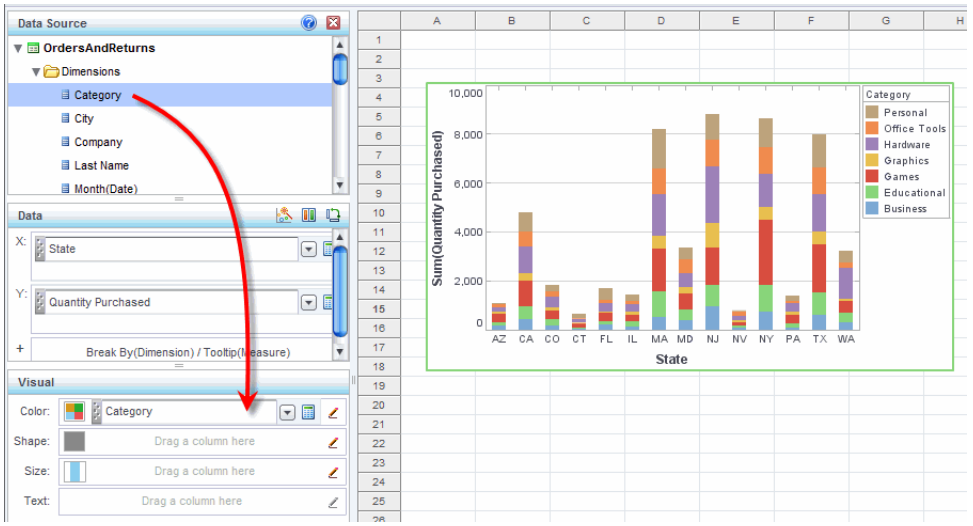
3. Drag the desired subseries dimension to one or more of the fields in the **Visual** panel at the bottom of the Chart Editor.

    a. Drag the dimension to the ‘Color’ field to discriminate the subgroups by color.

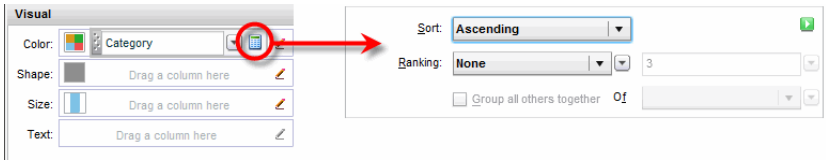
    b. Drag the dimension to the ‘Shape’ field to discriminate the subgroups by data point shape or fill pattern.

    c. Drag the dimension to the ‘Size’ field to discriminate the subgroups by the thickness of the representation element (bar, line, etc.).

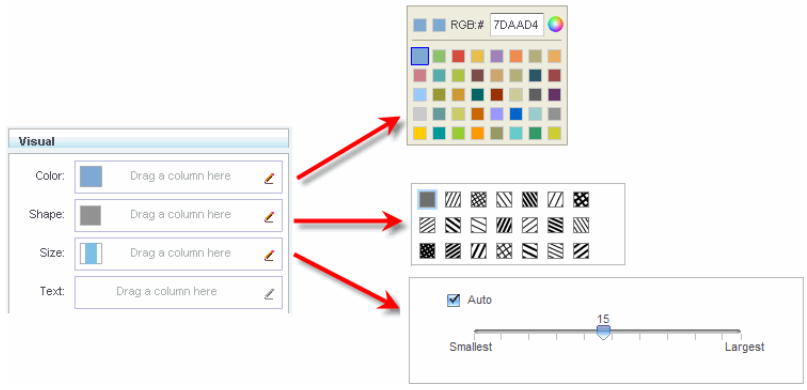
    d. Drag the dimension to the ‘Text’ field to discriminate the subgroups by placing appropriate text labels.



4. Click the ‘Edit Dimension’ button, and select the desired ordering and ranking for the coded values.



5. Click the appropriate ‘Edit’ button (‘Edit Color’, ‘Edit Shape’, ‘Edit Size’, ‘Edit Text’), and select the desired representation. For more information, see *Representing Data With Color*, *Representing Data With Shape*, *Representing Data with Size*, and *Representing Data With Text*.



Creating a sub-series automatically creates a corresponding legend. For information on controlling the legend display, see *Positioning a Chart Legend*. For information on further modifying the visual format of the subseries, see *Representing Data with Visual Formats*.

Example: Adding a Subseries

Assume that you want to plot 'Total Purchased' vs. 'State' for a dataset. Additionally, you want break down the sales in each region according to 'Category'. In this case, 'State' is the top level of grouping, represented by the labels on the X-axis, and 'Category' will be the subseries.

Follow the steps below:

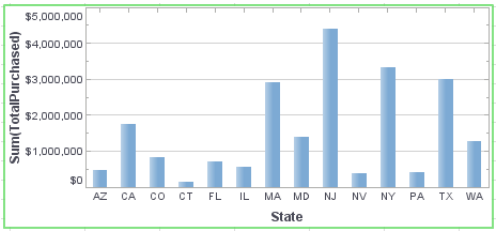
1. Create a new Viewsheet based on the sample 'Analysis' Worksheet.

2. Add a chart component, and click the 'Edit' button on the chart to open the Chart Editor.

3. In the 'Data Source' pane, expand the 'OrdersAndReturns' data block.

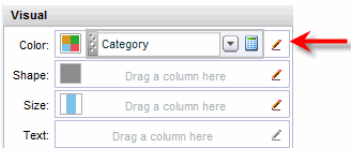
4. From the 'Dimensions' node in the **Data Source** panel, drag the 'State' dimension onto the 'X' field in the 'Data' panel.

5. From the 'Measures' node in the **Data Source** panel, drag the 'TotalPurchased' measure onto the 'Y' field in the **Data** panel.

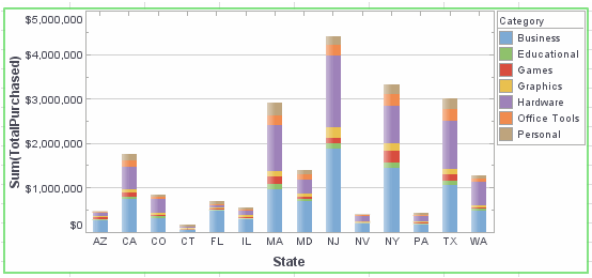


This displays 'Total Purchased' vs. 'State', as desired. You will now add the subseries.

6. From the 'Dimensions' node in the **Data Source** panel, drag the 'Category' dimension onto the 'Color' field in the **Visual** panel.



This creates the 'Category' subseries, discriminating the different categories using color, and automatically creates a corresponding legend. Resize the chart as desired to show all the data.



Notice that there are seven colors, one for each product category.

¢

See Also

Adding a Non-Formatted Subseries, for information on subseries without format.

Positioning a Chart Legend, for information on controlling the legend display.

Editing a Chart Legend, for information on changing the display of a legend.

Representing Data with Visual Formats, for information about ordering styles.

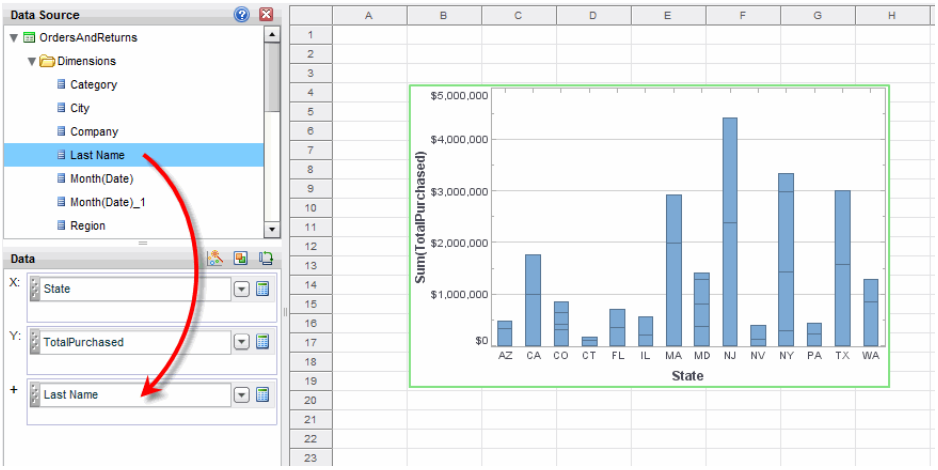
Adding a Non-Formatted Subseries

To add a subseries to an existing chart without visual formatting, follow the steps below:

1. Select the chart, and click the ‘Edit’ button to open the Chart Editor.

2. Expand the ‘Dimension’ node in the **Data Source** panel.

3. Drag the desired subseries dimension to the ‘Break By (Dimension) / Tooltip (Measure)’ field in the **Data** panel.



This creates a subseries without visual formatting and without a legend. This is useful when you want to represent an additional dimension on the chart without complicating the presentation. The example below provides an illustration.

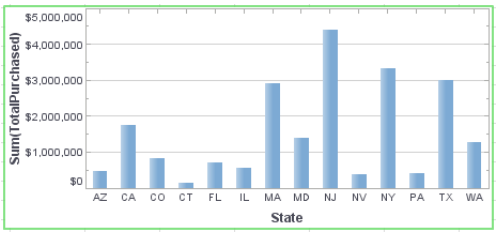
Example: Adding a Subseries

Assume that you want to plot 'Total Purchased' vs. 'State' for a dataset. Additionally, you want break down the sales in each region according to 'Date' (year). In this case, 'State' is the top level of grouping, represented by the labels on the X-axis, and 'Date' will be the subseries.

In addition to the 'Date' subseries, you would also like to see (for each state and each year) the names of the individuals who made those sales. You will present this information as a non-formatted subseries.

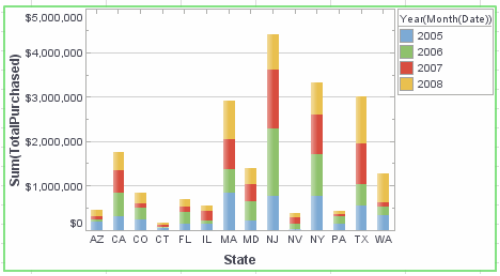
Follow the steps below:

1. Create a new Viewsheet based on the sample 'Analysis' worksheet.
2. Add a chart component, and click the 'Edit' button on the chart to open the Chart Editor.
3. In the **Data Source** panel, expand the 'OrdersAndReturns' data block.
4. From the 'Dimensions' node in the **Data Source** panel, drag the 'State' dimension onto the 'X' field in the **Data** panel.
5. From the 'Measures' node in the **Data Source** panel, drag the 'TotalPurchased' measure onto the 'Y' field in the **Data** panel.



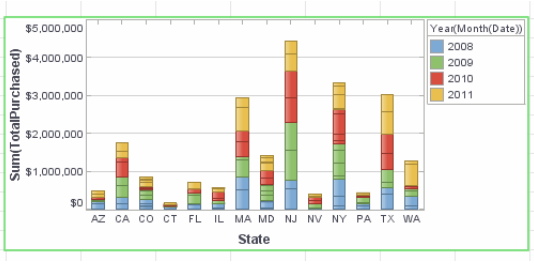
This displays ‘Total Purchased’ vs. ‘State’, as desired. You will now add the visually-formatted subseries.

6. From the ‘Dimensions’ node in the **Data Source** panel, drag the ‘Date’ dimension onto the ‘Color’ field in the **Visual** panel.



This creates the ‘Date’ subseries, formatting the different years using color, and creating a corresponding legend. (Resize the chart to show all the data.) In the next step, you will further break down the data by salesperson name.

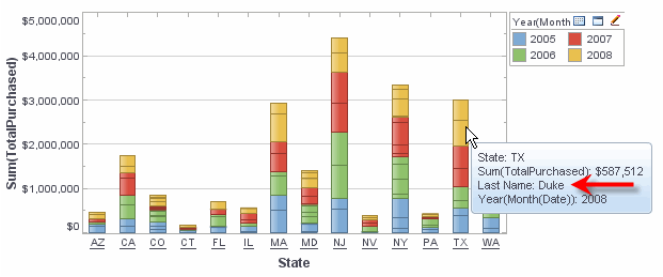
7. From the ‘Dimensions’ node in the **Data Source** panel, drag the ‘Last Name’ dimension onto the ‘Break By (Dimension) / Tooltip (Measure)’ field in the **Data** panel.



Note: You can also create an additional subseries by binding a vacant field in the ‘Visual’ panel. This will add another formatted subseries and legend.

This creates the ‘Last Name’ subseries without any additional formatting. Each ‘Year’ subgroup is now further broken down by salesperson name.

8. Preview the Viewsheet. Hover the mouse over the various chart elements to view the salesperson for each subgroup.



¢

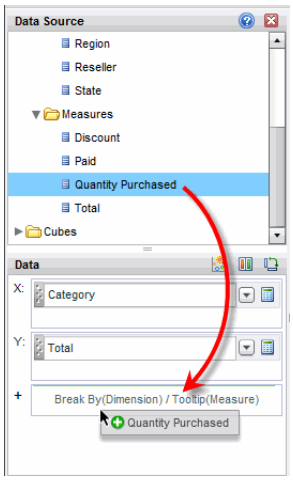
See Also

Adding a Visually-Formatted Subseries, for information on subseries with format.

Adding a Non-Displaying Measure to a Chart

In some cases, you may wish to add a non-displaying measure to a chart. This is useful when you want to utilize the measure for purposes of ranking or as part of a tooltip, but do not wish the measure to be displayed on the chart itself.

To add a non-displaying measure to a chart, simply drag the desired measure field to the 'Break By (Dimension) / Tooltip (Measure)' region. This will add the measure to the chart tooltip, and make the measure available for ranking.



See Also

Custom Tooltip, for information on customizing the text in a tooltip.
Editing a Dimension, for information on ranking dimension values.

3.4.8 Chart Properties

The Chart component's 'Properties' dialog box provides the following tabs: **General**, **Advanced**, **Hierarchy**, **Script**. The next sections discuss the chart-specific properties available under these tabs.

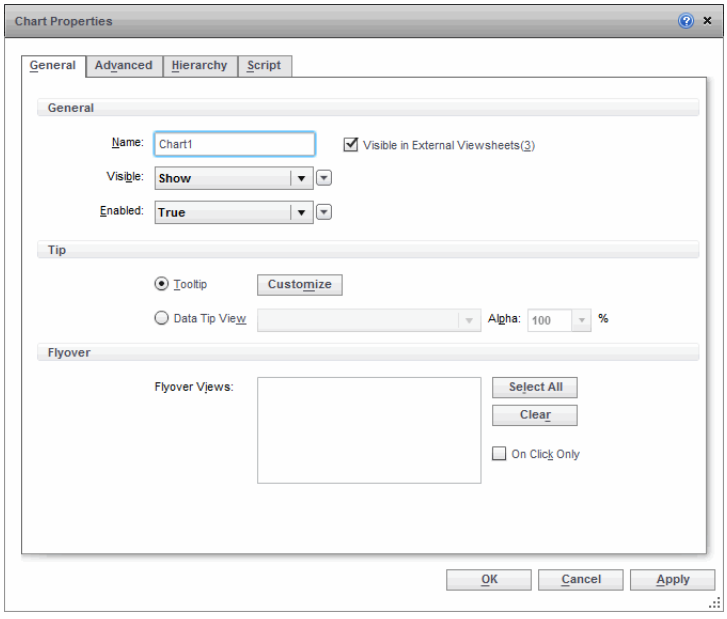
See Also
Properties Dialog Box, for information about common component properties.

Chart Properties: General Tab

The **General** tab in the 'Chart Properties' dialog box provides the following chart-specific properties.

Tip	**Tooltip**: Value is displayed as tooltip when mouse hovers over corresponding chart area. To display custom HTML in the tooltip, see *Custom Tooltip*. **Data Tip View**: A Viewsheet component is displayed when mouse hovers over chart area. The component is filtered based on the hover region. See *Data Tip View* for more details. **Flyover Views**: Allows you to select a set of Viewsheet components which will be adaptively filtered as the user hovers the mouse or clicks on the chart area. (Select 'On Click Only' if you want the filtered components to update when the user clicks.) Filtering is based on the group in the chart on which the user hovers or clicks. See *Flyover View* for more details.

See Also
General Tab, for information about common properties in the **General** tab.
Crosstab Properties, for information about the **General** tab.
Format Dialog Box, for a listing of format options.



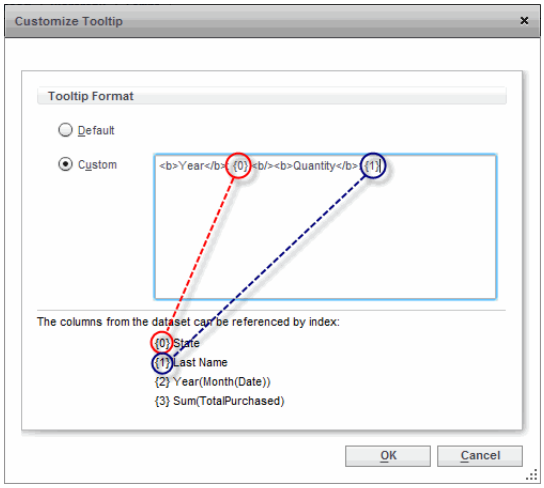
Custom Tooltip

You can create a custom tooltip by using HTML markup to style the tooltip text. The tooltip can incorporate data values in *java.text.MessageFormat* format.

To create a custom tooltip, follow the steps below:

1. Right-click the chart, and select 'Properties' from the context menu. This opens the 'Chart Properties' dialog box.
2. Click the **General** tab.
3. Select the 'Tooltip' option, and click the 'Customize' button. This opens the 'Customize Tooltip' dialog box.
4. Select the 'Custom' option, and enter the desired tooltip markup into the edit box.

The indices for the available columns are listed at the bottom of the dialog box.



You can use HTML tags to style the text (e.g., `<b>`, `<i>`, `<u>`, `<font>`) and to create line breaks (`<br/>`), etc. To insert data into the tooltip, enter the index of a data column within curly braces, e.g., `{1}`. Use *java.text.MessageFormat* format to format the inserted data. For example:

See *Format Tab* for information about format masks.

```
<b>Date</b>: {0, date, MMM-yy}
<b>Sales</b>: {1, number, $#,###.00}
```

To create even more complex text-based tooltips, you can use the *Data Tip View*, and specify a group of text elements as the data view assembly to display.

Example:
Custom Tooltip

In this example, you will create a chart that displays quantity purchased by company. When the user hovers the mouse over a given company (e.g., "Rutgers Bank"), the Chart will display a tooltip similar to the following:

```
Rutgers Bank
27 units purchased
```

Follow the steps below:

1. Create a new Viewsheet based on the sample 'US Sales' Worksheet.

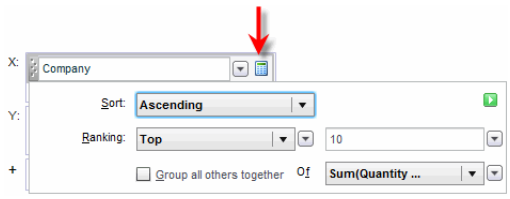
2. Drag a Chart element onto the Viewsheet grid.

3. Click the center of the Chart (or click the 'Edit' button at the top-right). This opens the Chart Editor. Make the following selections:

   a. From the 'Dimensions' node in the **Data Source** panel, drag the 'Company' dimension to the 'X' field in the **Data** panel.

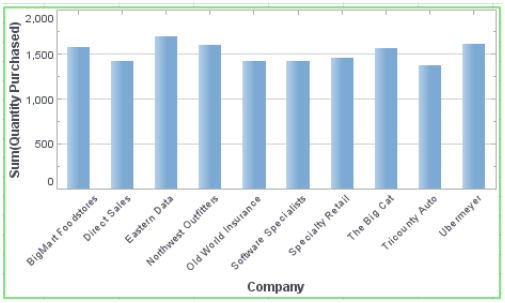
   b. From the 'Measures' node in the **Data Source** panel, drag the 'Quantity Purchased' measure to the 'Y' field in the **Data** panel. This creates a chart that displays quantity purchased for each company.

   c. Click the 'Edit Dimension' button, and select 'Ranking' to display only the top 10 companies. Click the 'Apply' button.



   d. Right-click on the one of the Chart's X-axis labels, and select 'Axis Properties' from the context menu. Choose a slanted text rotation and a smaller font size to improve the axis appearance. Click 'OK'.

   e. Resize the chart as desired.



4. Right-click the Chart, and select ‘Properties’ from the context menu. This opens the ‘Chart Properties’ dialog box.

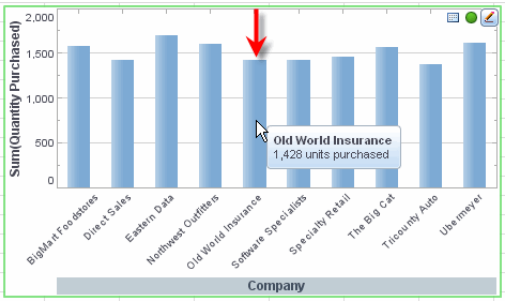
   a. Under the **General** tab, select the ‘Tooltip’ option, and click ‘Custom’. This opens the ‘Customize Tooltip’ dialog box.

   b. Select the ‘Custom’ option and enter the following in the text field:

   ```
   <b>{0}</b><br />{1} units purchased
   ```

   c. Click ‘OK’ to close the ‘Customize Tooltip’ dialog box, and click ‘OK’ again to close the ‘Chart Properties’ dialog box.

When you hover the mouse over the bars on the chart, you will now see the custom tooltip you specified.



¢

See Also

Data Tip View, to create complex tooltips using Viewsheet components.
Adding a Non-Displaying Measure to a Chart, to utilize a measure in the tooltip which is not displayed on the chart.

Data Tip View

For a Table data-tip, enable ‘Shrink to Fit’ in the *Table Properties* to minimize the data-tip footprint.

A *Data Tip* displays information about the current chart or table region using another Viewsheet component, such as another chart, text, Gauge, or *Organization Components*. The data tip is displayed as a tooltip (next to the

pointer), and is adaptively filtered to show only data corresponding to the particular region of the chart or crosstab where the pointer is positioned. If the data tip utilizes a table or chart, the export features of the table or chart remain available in data tip view.

Walkthrough

In this example, you will create a chart that displays quantity purchased by company. When the user hovers the mouse over a given company, the Chart will display a Gauge as a Data Tip. This Gauge will display the average quantity purchased by the individual company.

1. Create a new Viewsheet based on the sample 'US Sales' Worksheet.

2. Drag a Gauge from the Component tree into the Viewsheet.

3. Configure the Gauge to display the average quantity purchased:

    a. From the 'Sales' Data Block (top of the Component tree), drag the 'Quantity Purchased' field onto the Gauge. This binds the Gauge to the Data Block.

    b. Right-click the Gauge, and select 'Gauge Properties' from the context menu. This opens the 'Properties' dialog box for the Gauge.

    c. Under the **General** tab, change the 'Name' property to "QuantityGauge".

    d. Set the following limits: 'Maximum'=200, 'Minor Increment'=10, 'Major Increment'=25.

    e. Under the **Data** tab, select 'Count' form the 'Aggregate' menu. This will display the number of records (orders) for each company.

    f. Click 'OK' to close the 'Gauge Properties' dialog box.

    g. Resize the Gauge to be about half the default size.

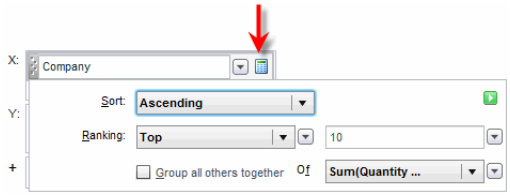
4. Drag a Chart element onto the Viewsheet grid.

5. Click the center of the Chart (or click the 'Edit' button at the top-right). This opens the Chart Editor. Make the following selections:

    a. From the 'Dimensions' node in the **Data Source** panel, drag the 'Company' dimension to the 'X' field in the **Data** panel.

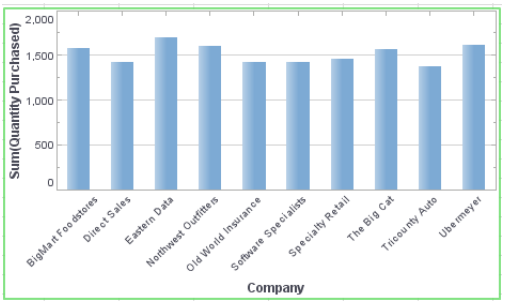
b. From the ‘Measures’ node in the **Data Source** panel, drag the ‘Quantity Purchased’ measure to the ‘Y’ field in the **Data** panel. This creates a chart that displays quantity purchased for each company.

c. Click the ‘Edit Dimension’ button, and select ‘Ranking’ to display only the top 10 companies. Click the ‘Apply’ button.



d. Right-click on the one of the Chart’s X-axis labels, and select ‘Axis Properties’ from the context menu. Choose a slanted text rotation and a smaller font size to improve the axis appearance. Click ‘OK’.

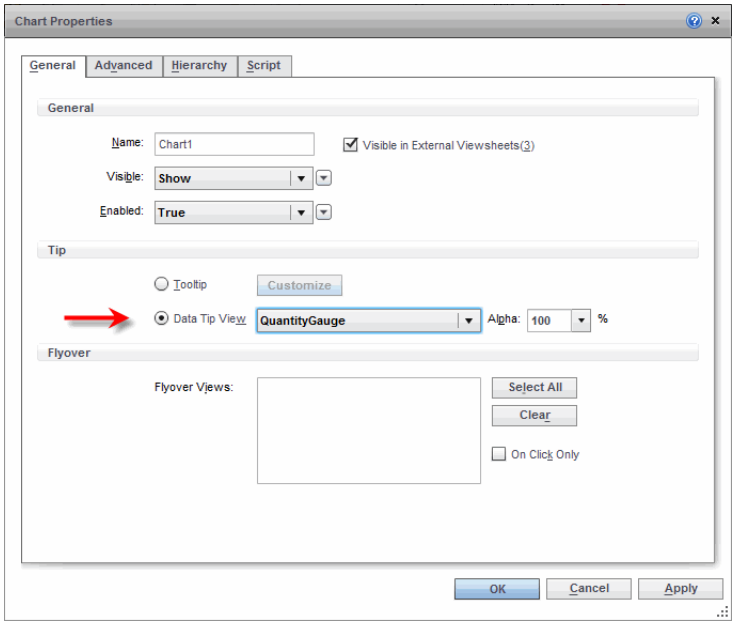
e. Resize the chart as desired.



6. Right-click the Chart, and select ‘Properties’ from the context menu. This opens the ‘Chart Properties’ dialog box.

a. Under the **General** tab, select ‘Data Tip View’. From the adjacent menu, select the ‘QuantityGauge’ component.

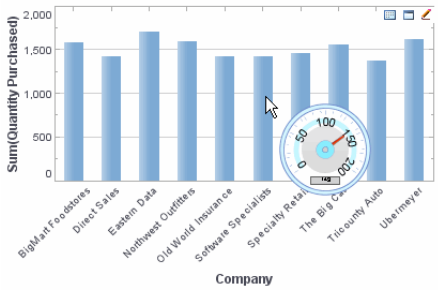
b. To adjust the transparency of the data tip component (Gauge), enter a value in the ‘Alpha’ menu (or choose one of the preset values). A value of 0 indicates complete transparency; a value of 100% indicates complete opacity.



c. Click ‘OK’ to close the ‘Chart Properties’ dialog box.

Note: Data tips are not active in the Design view. Preview the Viewsheet to see them.

7. Preview the Viewsheet, and hover the mouse over different companies in the Chart to view the corresponding Data Tip (i.e., number of orders).



Note that the Gauge is *not* displayed as part of the Viewsheet layout. The Gauge appears only when you hover the mouse over one of the companies in the chart, and it then displays only the data for that particular company.

You can use a single component in both a Data Tip view and a *Flyover View*. If you designate a Data Tip component for Flyover duty, the component always remains visible on the Viewsheet.

See Also

Flyover View, to adaptively filter Viewsheet components based on mouse position.

Flyover View

A *flyover* filters the information displayed in selected Viewsheet components when the user moves the mouse over (or clicks on) particular regions of a chart, table, or crosstab. This effectively allows a Chart, Table, or Crosstab to behave as a selection element. Flyover is similar to the *Data Tip View*, except that the filtered components remain in their original positions (rather than following the mouse).

Walkthrough

In this example, you will create a chart that displays quantity purchased by company. When the user click a given company, the Chart will display a Gauge as a Data Tip. This Gauge will display the average quantity purchased by the individual company.

1. Create a new Viewsheet based on the sample 'US Sales' Worksheet.

2. Drag a Gauge from the Component tree into the Viewsheet.

3. Configure the Gauge to display the average quantity purchased. Follow these steps:

    a. From the 'Sales' Data Block (top of the Component tree), drag the 'Quantity Purchased' field onto the Gauge. This binds the Gauge to the Data Block.

    b. Right-click the Gauge, and select 'Properties' from the context menu. This opens the 'Properties' dialog box for the Gauge.

    c. Under the **General** tab, change the 'Name' property to "QuantityGauge".

    d. Set the following limits: 'Maximum'=200, 'Minor Increment'=10, 'Major Increment'=25.

    e. Under the **Data** tab, select 'Count' form the 'Aggregate' menu. This will display the number of records (orders) for each company.

    f. Click 'OK' to close the 'Gauge Properties' dialog box.

4. Drag a Table component onto the Viewsheet grid. Follow the steps below to configure the Table:

    a. In the 'Sales' Data Block (top of the Component tree), select the following fields: 'Category', 'Company', 'Name'.

b. Drag the selected fields onto the Table component. This adds the three columns to the Table. Resize the Table as desired.

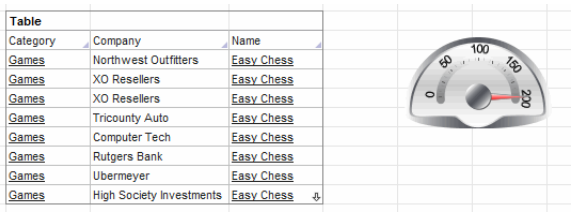


Table		
Category	Company	Name
Games	Northwest Outfitters	Easy Chess
Games	XO Resellers	Easy Chess
Games	XO Resellers	Easy Chess
Games	Tricounty Auto	Easy Chess
Games	Computer Tech	Easy Chess
Games	Rutgers Bank	Easy Chess
Games	Ubermeyer	Easy Chess
Games	High Society Investments	Easy Chess

c. Right-click the Table, and select ‘Properties’ from the context menu. This opens the ‘Table Properties’ dialog box.

d. Under the **General** tab, change the ‘Name’ property to “CompanyTable”.

e. Under the **Advanced** tab, select ‘Shrink to Fit’. This automatically sizes the table to the number of records displayed.

f. Press ‘OK’ to close the dialog box.

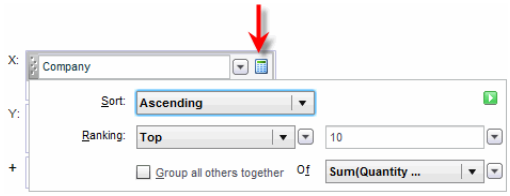
5. Drag a Chart element onto the Viewsheet grid.

6. Click the center of the Chart (or click the ‘Edit’ button at the top-right). This opens the Chart Editor. Make the following selections:

a. From the ‘Dimensions’ node in the **Data Source** panel, drag the ‘Company’ dimension to the ‘X’ field in the **Data** panel.

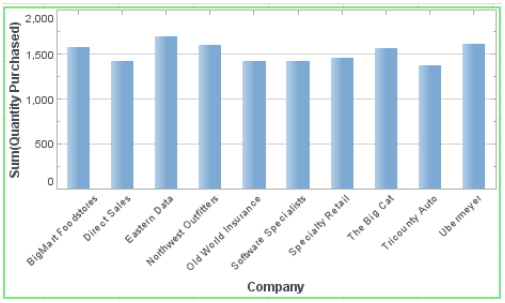
b. From the ‘Measures’ node in the **Data Source** panel, drag the ‘Quantity Purchased’ measure to the ‘Y’ field in the **Data** panel. This creates a chart that displays quantity purchased for each company.

c. Click the ‘Edit Dimension’ button, and select ‘Ranking’ to display only the top 10 companies. Click the ‘Apply’ button.



d. Right-click on the one of the Chart’s X-axis labels, and select ‘Axis Properties’ from the context menu. Choose a slanted text rotation and a smaller font size to improve the axis appearance. Click ‘OK’.

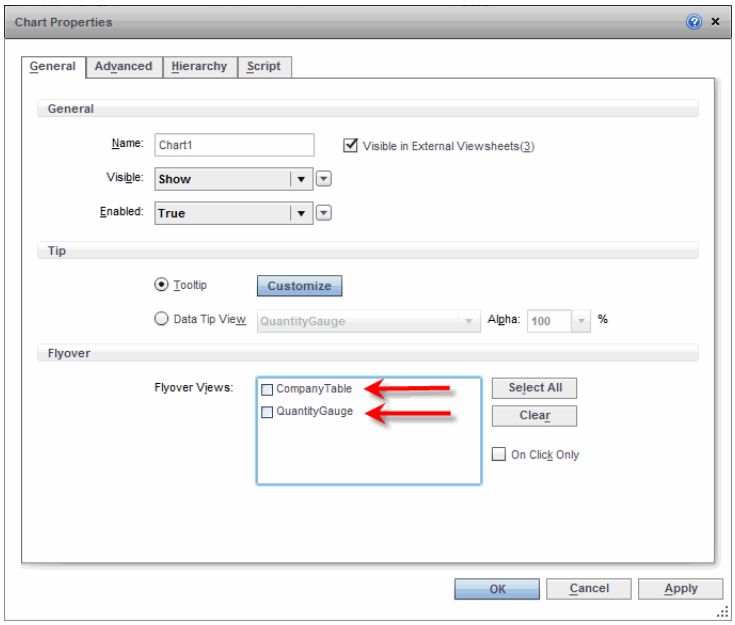
e. Resize the chart as desired.



7. Right-click the Chart, and select ‘Properties’ from the context menu. This opens the ‘Chart Properties’ dialog box.

   a. Select the **General** tab.

   b. In the ‘Flyover’ panel, click to select the two available components: ‘CompanyTable’ and ‘QuantityGauge’.

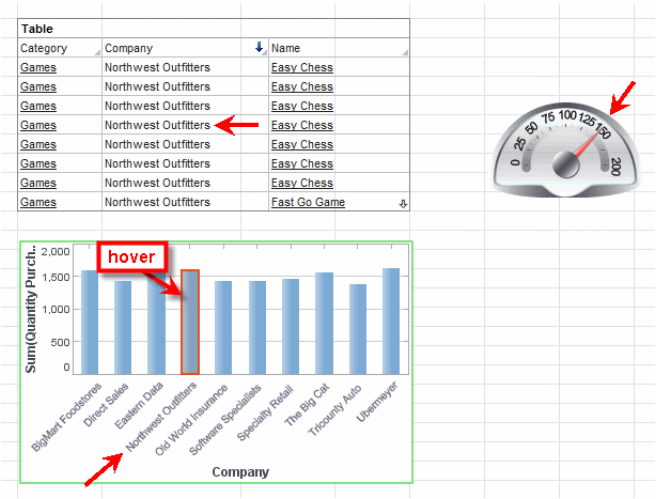


   c. (Optional) Select ‘On Click Only’ if you want the filtered components (‘CompanyTable’ and ‘QuantityGauge’) to update only when the user clicks on the chart. Otherwise, the components will update whenever the user hovers the mouse over the chart.

   d. Click ‘OK’ to close the ‘Chart Properties’ dialog box.

Note: Data tips are not active in the Design view. Preview the Viewsheet to see them.

8. Preview the Viewsheet, and hover the mouse (or click) over different companies in the Chart. Observe that both the Table and Gauge are filtered to display only the corresponding data.



9. To lock the selection to a particular company (or set of companies), click on the Chart to select the desired company (or Ctrl-click to select a set of companies). This will lock the filter in place until the selection is removed.

You can use a single component in both a *Data Tip View* and a Flyover view. If you designate a Data Tip component for Flyover duty, the component always remains visible on the Viewsheet.

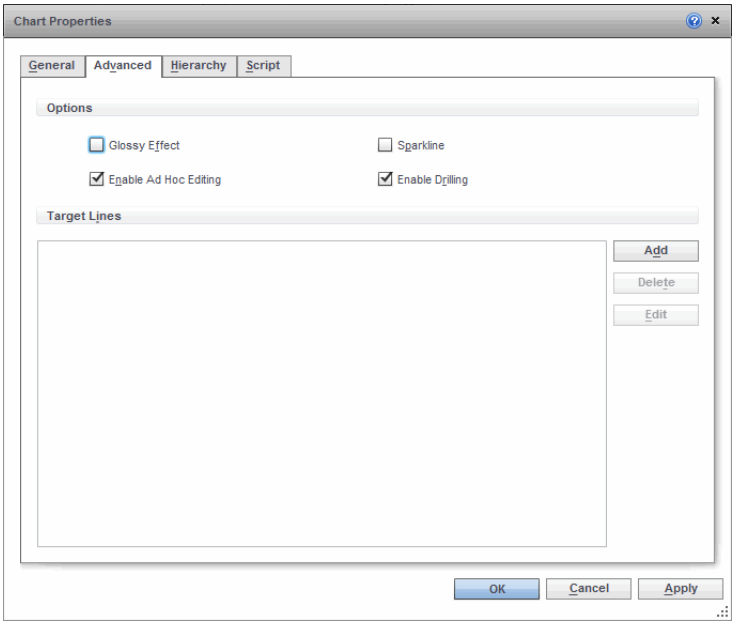
See Also

Brushing a Chart, to use Chart selections to highlight (rather than filter) other components.

Chart Properties: Advanced Tab

The **Advanced** tab in the 'Chart Properties' dialog box provides the following chart-specific properties.

Apply Effect	Applies a 3D effect to the Chart as a whole.
Enable Ad Hoc Editing	Allow end users to access the Chart Editor and edit the chart.
Sparkline	Hides all axes and legends. For a line chart, the 'Sparkline' also adds a point to the end of the line, and for a bar chart, displays negative values in red.
Enable Drilling	Allows the user to drill up and down in date or dimension fields. See *Drilling Down into a Chart* for more information.
Target Lines	Specify marker lines, bands, or statistical regions to be placed in the plot area. See *Adding a Target Line*, *Adding a Target Band*, and *Adding a Statistical Measure* for more information.

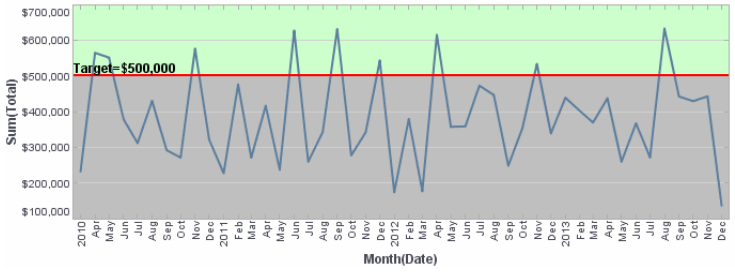


See Also

Editing an Axis Title, for information about altering title text.
Editing a Chart Axis, for information about hiding chart axes.

Adding a Target Line

A *target line* is a horizontal or vertical line drawn on the chart that generally denotes either an ideal value (goal or threshold) or representative value (average, minimum, etc.). Regions above and below the target value can be assigned independent colors.



To add a target line, follow the steps below:

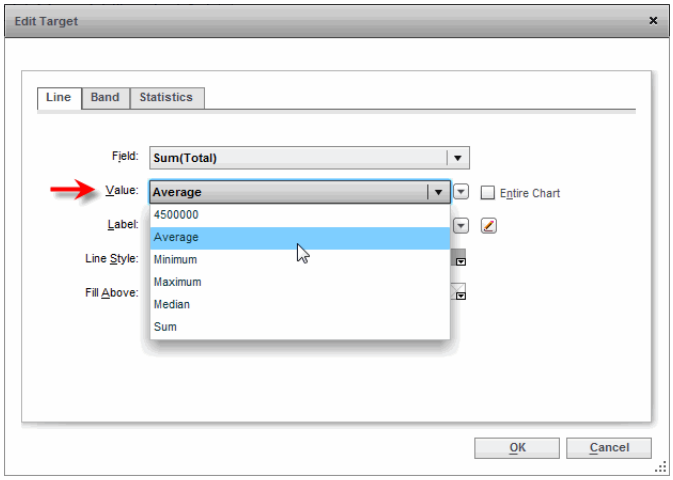
1. Right-click the Chart, and select ‘Properties’ from the context menu. This opens the ‘Chart Properties’ dialog box.

2. Select the **Advanced** tab of the ‘Properties’ dialog box. In the ‘Target Lines’ panel, press the ‘Add’ button. This opens the ‘Add Target’ dialog box.

3. Select the **Line** tab.

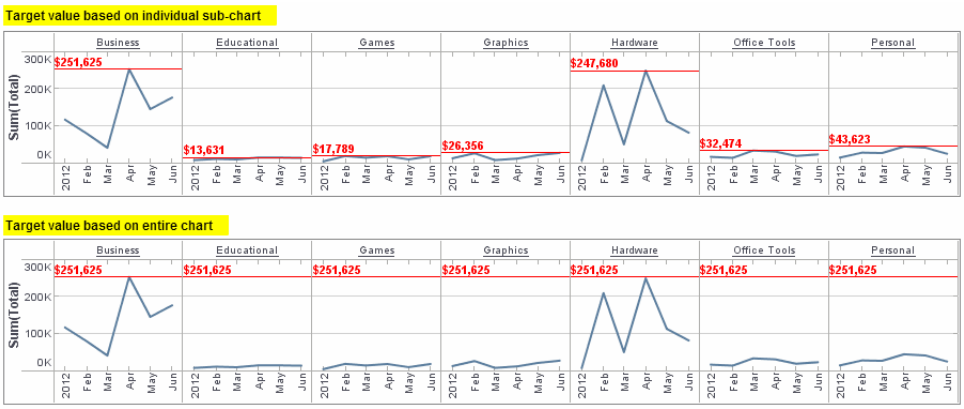
4. In the ‘Field’ menu, select the chart measure to which you want to add the target line.

5. In the ‘Value’ field, type a numerical value at which to place the target line for the selected measure, or choose one of the following options to compute the target value from the data: ‘Average’, ‘Minimum’, ‘Maximum’, ‘Median’, ‘Sum’. (For example, select ‘Average’ to place the target line at the average value of the selected measure.)

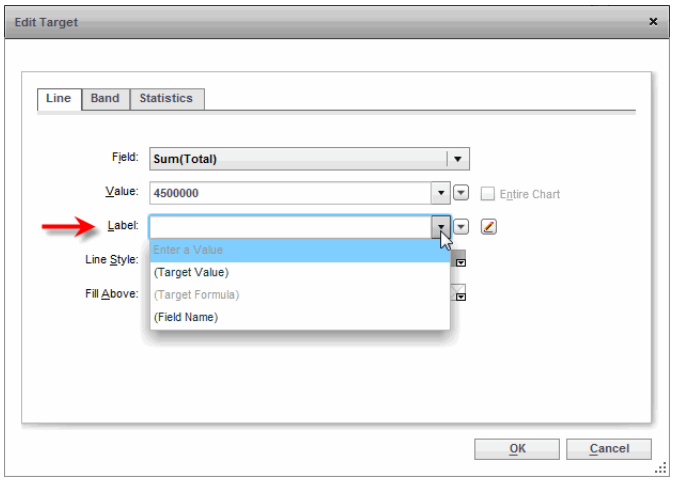


6. (Optional) If you select one of the available target line computations (‘Average’, ‘Minimum’, etc.), enable the ‘Entire Chart’ option to compute the target value based on measure data from the entire chart. Disable the ‘Entire Chart’ option to compute the target value for each sub-chart based only on measure data from the same sub-chart.

   The following illustration demonstrates the effect of the ‘Entire Chart’ setting (‘Value’ is set to ‘Maximum’ in both cases).



7. From the ‘Label’ menu, select one of the following label options:



   a. Select ‘Enter a Value’ to type a custom label for the target line.

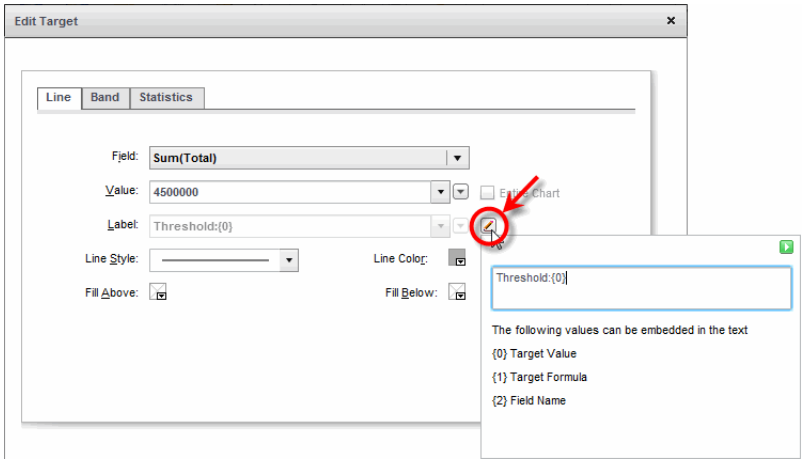
   b. Select ‘(Target Value)’ to insert the numerical value of the target line as the label.

   c. Select ‘(Target Formula)’ to insert the name of the computation method (‘Average’, ‘Minimum’, etc.) as the label, if applicable.

   d. Select ‘(Field Name)’ to insert the field name for the selected measure as the label; for example, “Sum(Total)”.

Note: The customized label supersedes any previous selection from the ‘Label’ menu.

8. (Optional) To further customize the label, press the ‘Edit’ button next to the ‘Label’ field. This opens a panel in which you can manually enter the label. Press the green ‘Apply’ button when you have finished entering the label.

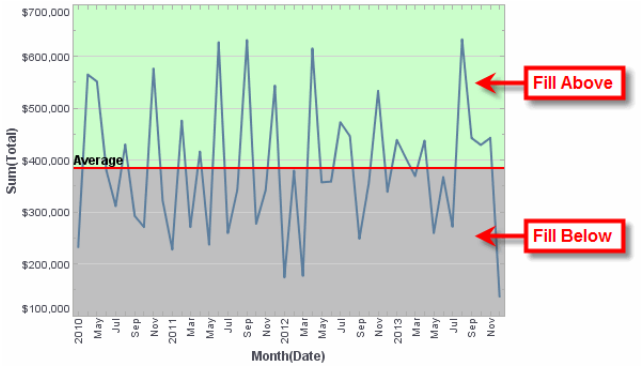


If desired, you can add the target value, target formula, and field name into the label by inserting the corresponding codes (`{0}`,`{1}`,`{2}`) shown at the bottom of the panel. You can apply formats to the inserted values by using the same syntax for *Custom Tooltip*. Some examples are shown below:

```
{1} = {0,number,$#,##0}    yeilds    “Average = $383,485”
{1} of monthly {2}         yeilds    “Average of monthly
Sum(Total)”
```

9. Select a ‘Line Style’ and ‘Line Color’ in which to display the target line.

10. (Optional) Select a ‘Fill Above’ color and ‘Fill Below’ color to fill the regions of the chart above and below the target line respectively.



11. Press ‘OK’ to close the dialog box.

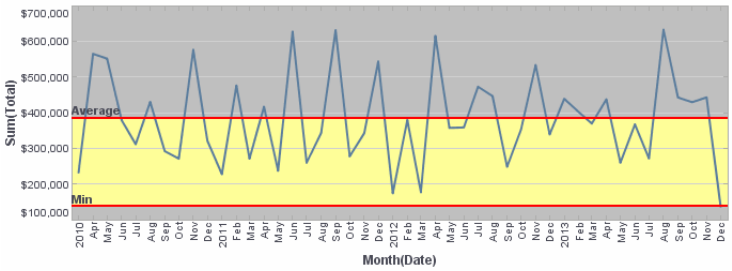
By default, the target line appears on the chart even if its value is greater than the largest data point. This may sometimes cause the data points on the chart to be compressed into a small region of the plot area, which makes the chart difficult to read. To correct this, turn off the ‘Keep Element in Plot’ option in the ‘Plot Properties’ dialog box. See *Editing Plot Properties* for more information.

See Also

Adding a Target Band, to demarcate a specified data region on the chart.
Adding a Statistical Measure, to display statistical measures on the chart.
Adding Property Script (Expressions), in *Advanced Dashboard Scripting*, to set target line values and labels through expressions.
Dynamic Properties, set target line values and labels through *Input Components*.

Adding a Target Band

A *target band* is a horizontal or vertical band drawn on the chart that generally denotes either an ideal range (e.g., goal zone) or representative range (e.g., span of maximum to minimum). The region within the target band, as well as the regions above and below, can be assigned independent colors.



To add a target band, follow the steps below:

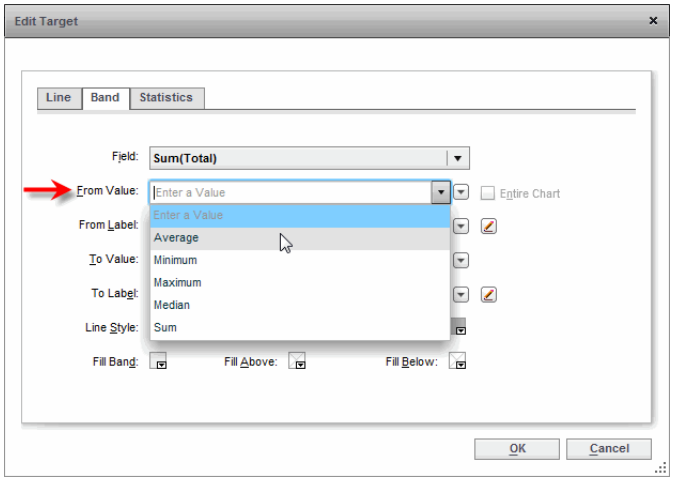
1. Right-click the Chart, and select ‘Properties’ from the context menu. This opens the ‘Chart Properties’ dialog box.

2. Select the **Advanced** tab of the ‘Properties’ dialog box. In the ‘Target Lines’ panel, press the ‘Add’ button. This opens the ‘Add Target’ dialog box.

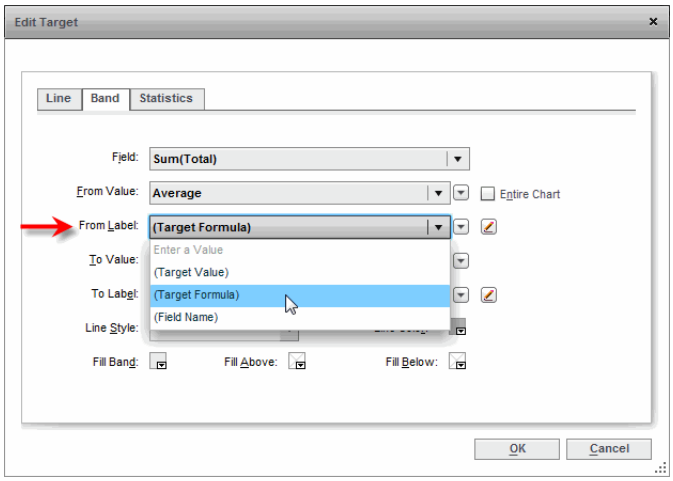
3. Select the **Band** tab.

4. In the ‘Field’ menu, select the chart measure to which you want to add the target band.

5. In the ‘From Value’ field, enter a numerical value at which to place the lower band range for the selected measure, or choose one of the following options to compute the lower band range from the data: ‘Average’, ‘Minimum’, ‘Maximum’, ‘Median’, ‘Sum’. (For example, select ‘Average’ to place the lower band boundary at the average value of the selected measure.)



6. In the ‘From Label’ menu, select one of the following label options:



   a. Select ‘Enter a Value’ to type a custom label for the lower band boundary.

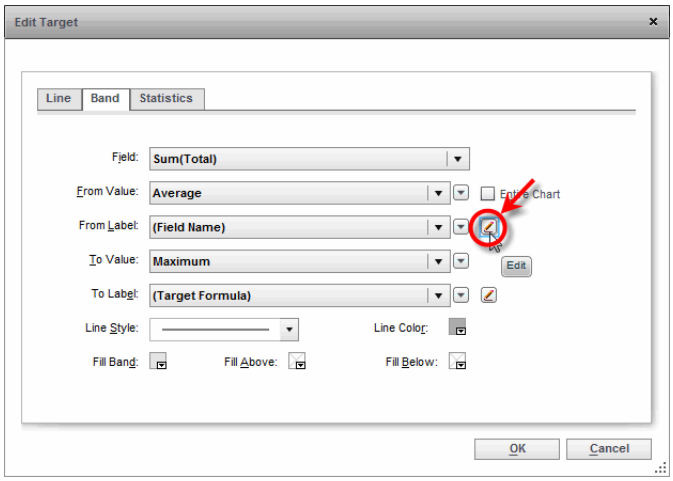
   b. Select ‘(Target Value)’ to insert the numerical value of the lower band boundary as the label.

   c. Select ‘(Target Formula)’ to insert the name of the computation method (‘Average’, ‘Minimum’, etc.) as the lower band boundary label, if applicable.

   d. Select ‘(Field Name)’ to insert the field name of the selected measure as the lower band boundary label, e.g., “Sum(Total)”.

Note: The customized label supersedes any previous selection from the ‘From Label’ menu.

7. (Optional) To further customize the label, press the ‘Edit’ button next to the ‘From Label’ field. This opens a panel in which you can manually enter the label. Press the green ‘Apply’ button when you have finished entering the label.

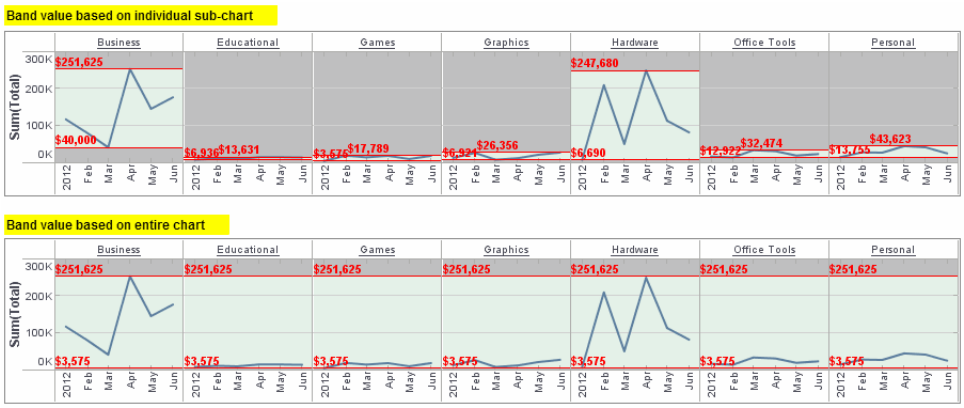


If desired, you can add the target band value, target band formula, and field name into the label by inserting the corresponding codes (`{0},{1},{2}`) shown at the bottom of the panel. You can format the inserted values using the same syntax as *Custom Tooltip*. Some examples are shown below:

```
{1} = {0,number,$#,##0}   yeilds   “Average = $383,485”
{1} of monthly {2}        yeilds   “Average of monthly
Sum(Total)”
```

8. (Optional) If you select one of the available target band computations (‘Average’, ‘Minimum’, etc.), enable the ‘Entire Chart’ option to compute the target band value based on measure data from the entire chart. Disable the ‘Entire Chart’ option to compute the target band value for each sub-chart based only on measure data from the same sub-chart.

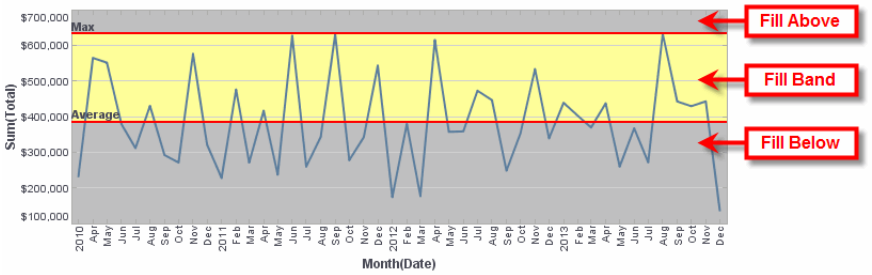
   The following illustration demonstrates the effect of the ‘Entire Chart’ setting. (In both cases, ‘From Value’ is set to ‘Minimum’ and ‘To Value’ is set to ‘Maximum’).



9. Repeat the previous steps to set the ‘To Value’ and ‘To Label’ properties, which specify the position and label of the upper band boundary.

10. Select a ‘Line Style’ and ‘Line Color’ in which to display the upper and lower target band boundaries.

11. (Optional) Press the ‘Fill Band’ button and select a background color to fill the band between the lower and upper boundaries. Select a ‘Fill Above’ color and ‘Fill Below’ color to fill the regions of the chart above and below the band boundaries, respectively.



12. Press ‘OK’ to close the dialog box.

By default, the target band appears on the chart even if its upper or lower range values are greater than the largest data point. This may sometimes cause the data points on the chart to be compressed into a small region of the plot area, which makes the chart difficult to read. To correct this, turn off the ‘Keep Element in Plot’ option in the ‘Plot Properties’ dialog box. See *Editing Plot Properties* for more information.

See Also

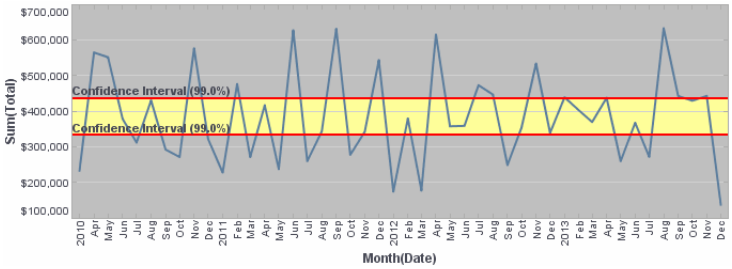
Adding a Target Line, to place a target line or representative line on the chart.

Adding a Statistical Measure, to display statistical measures on the chart.

Adding Property Script (Expressions), in *Advanced Dashboard Scripting*, to set target line values and labels through expressions.
Dynamic Properties, set target line values and labels through *Input Components*.

Adding a Statistical Measure

A *statistical measure* is represented by one or more lines drawn on the chart to indicate the values of statistical quantities derived from the data (confidence intervals, percentages, percentiles, quantiles, or standard deviation).



To add a statistical measure, follow the steps below:

1. Right-click the Chart, and select ‘Properties’ from the context menu. This opens the ‘Chart Properties’ dialog box.

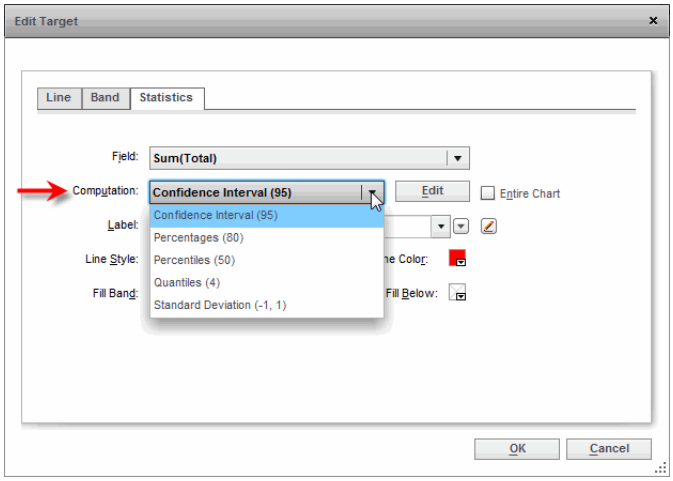
2. Select the **Advanced** tab of the ‘Properties’ dialog box.

3. In the ‘Target Lines’ panel, press the ‘Add’ button. This opens the ‘Add Target’ dialog box.

4. Select the **Statistics** tab.

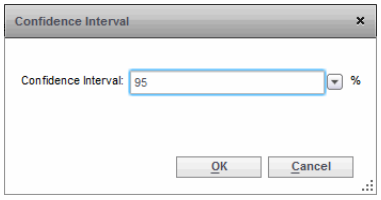
5. In the ‘Field’ menu, select the chart measure to which you want to add the statistical measure.

6. In the ‘Computation’ field, select one of the following options to compute statistics from the data: ‘Confidence Interval’, ‘Percentages’, ‘Percentile’, ‘Quantiles’, ‘Standard Deviation’. (See explanations below.)

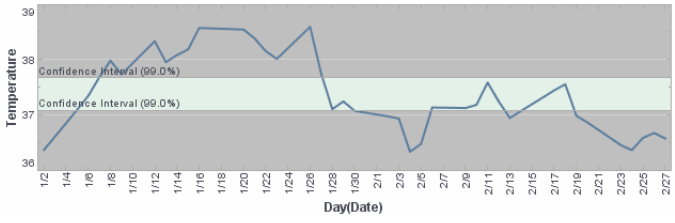


7. To modify the statistical measure, press the ‘Edit’ button. The following settings are available:

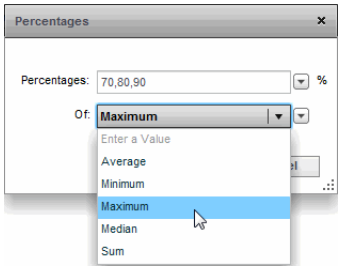
   a. **Confidence Interval:** For the ‘Confidence Interval’ option, enter a value as a percentage.



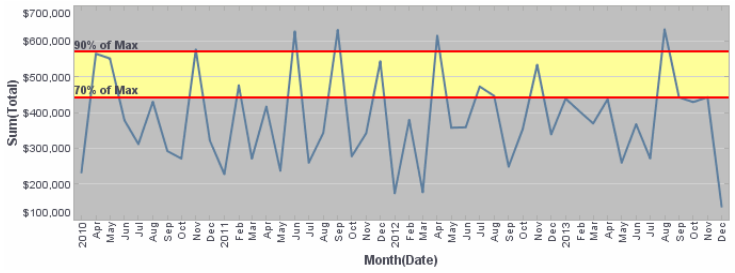
   The resulting top and bottom confidence bounds indicate the interval of values in which the “true” value is expected to be found. For example, the “true” temperature in the chart below would be expected to fall within the displayed confidence interval in 99 out of 100 such samples. (In other words, the true temperature is expected to be *outside* the confidence bounds purely by chance in 1 out of 100 samples.)



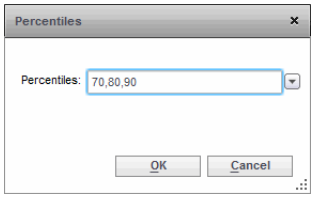
   b. **Percentages:** For the ‘Percentages’ option, enter a value or comma-separated list of values as percentages. In the ‘Of’ field, specify the basis on which the percentage should be computed. You can type a fixed value or select from the following presets: ‘Average’, ‘Minimum’, ‘Maximum’, ‘Median’, ‘Sum’.



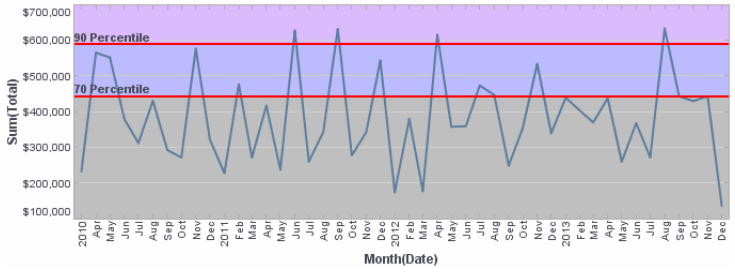
For example, to display percentage lines at 70% and 90% of the Maximum, enter "`70,90`" in the 'Percentages' field and select the 'Maximum' option from the 'Of' field.



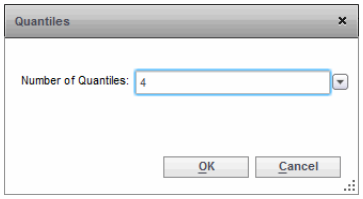
c. **Percentiles:** For the 'Percentiles' option, enter a value or comma-separated list of values as percentages.



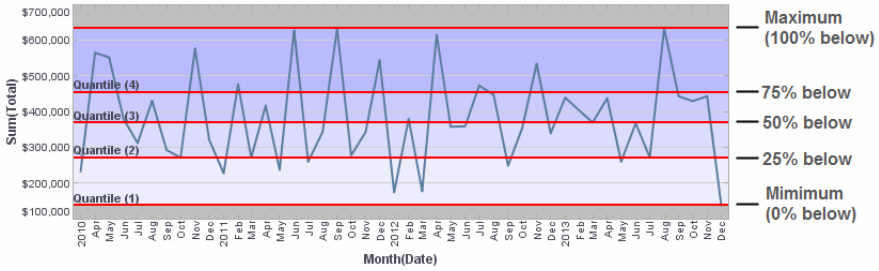
The resulting percentile lines indicate the levels below which the specified percentages of values are found. For example, percentile lines at 70% and 90% ("`70,90`" in the 'Percentages' field) designate the levels, respectively, below which 70% and 90% of the data are found.



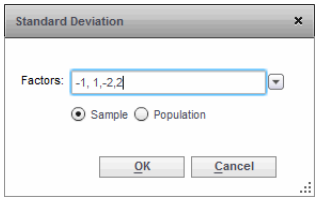
d. **Quantiles:** For the 'Quantiles' option, enter the number of quantiles to display.



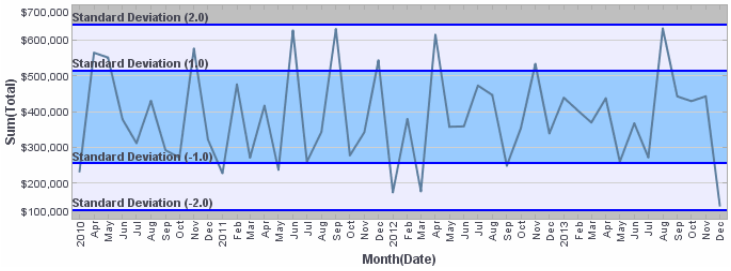
The resulting quantile lines are evenly distributed between 0% and 100% and indicate the levels below which the specified percentage of values are found. For example, enter "4" as the 'Number of Quantiles' to generate lines designating the levels below which 25%, 50%, and 75% of the data are found. This creates four regions in the data: 0-25%, 25%-50%, 50%-75%, and 75%-100%.



e. **Standard Deviation:** For the 'Standard Deviation' option, enter a comma-separated list of factors. Each successive pair of factors represents, respectively, the lower and upper multipliers for the standard deviation.



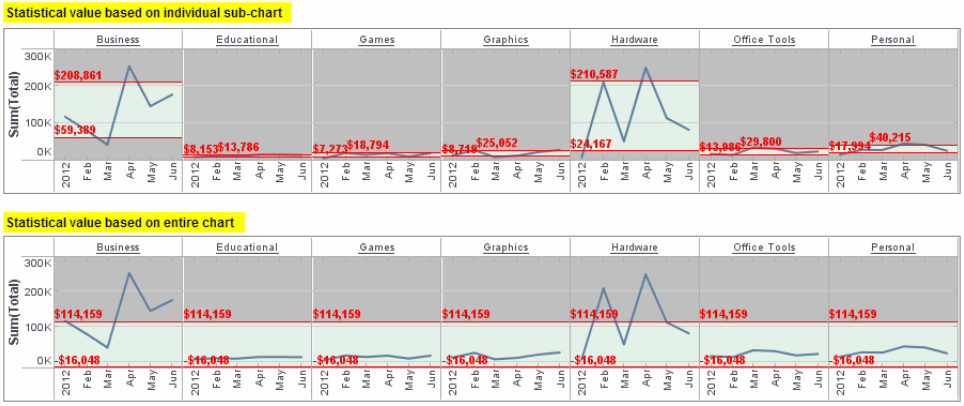
For example, enter "`-1,1,-2,2`" in the 'Factors' field to draw lines, respectively, at 1 standard deviation below the mean, 1 standard deviation above the mean, 2 standard deviations below the mean, and 2 standard deviations above the mean.



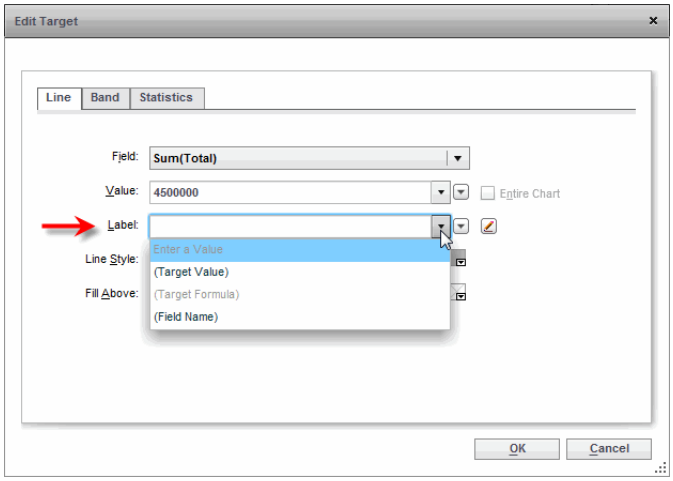
Select the 'Sample' option to compute the sample standard deviation or select the 'Population' option to compute the population standard deviation. (The distinction between sample and population standard deviation can be found in any statistical reference.)

8. (Optional) Enable the 'Entire Chart' option to compute the statistical values based on measure data from the entire chart. Disable the 'Entire Chart' option to compute the statistical values for each sub-chart based only on measure data from the same sub-chart.

   The following illustration demonstrates the effect of the 'Entire Chart' setting ('Computation' is set to 'Standard Deviation' in both cases).

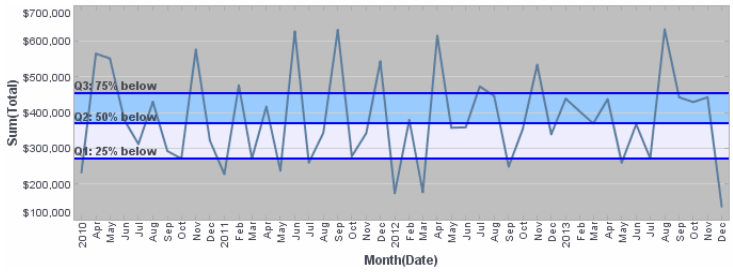


9. From the 'Label' menu, select one of the following label options:



To enter a literal **comma** in the label, escape the comma with a backslash (e.g., "Q1\,25% below, Q2\, 50% below, Q3\,75% below").

a. Select 'Enter a Value' to type custom labels for the statistical measures. Labels for individual lines should be separated by commas. For example, if you are generating the 4-quantile (which creates three lines), enter three labels separated by commas, e.g., "`Q1: 25% below, Q2: 50% below, Q3: 75% below`".



If you enter only a single label, this label will be attached to all of the lines. This can be useful when you include customization codes in the label, as described below.

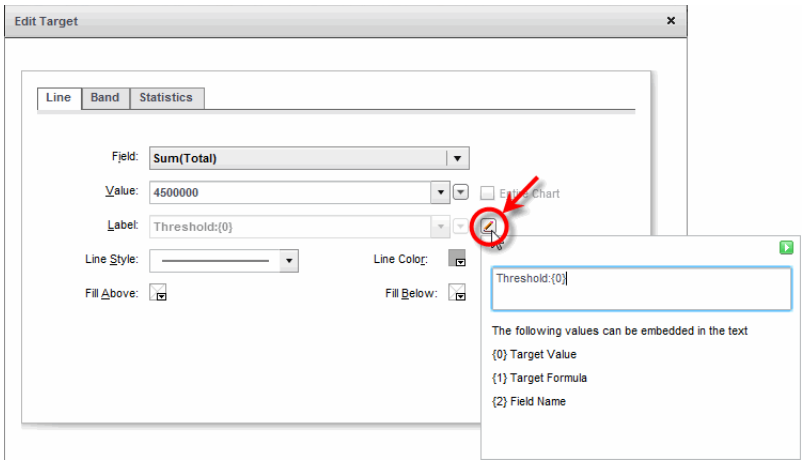
b. Select '(Target Value)' to insert the numerical value of the lines as the labels.

c. Select '(Target Formula)' to insert the name of the computation method (e.g., 'Quantile 1', 'Quantile 2', etc.) as the label, if applicable.

d. Select '(Field Name)' to insert the field name for the selected measure as the label, e.g., "Sum(Total)".

Note: The customized label supersedes any previous selection from the ‘Label’ menu.

10. (Optional) To further customize the labels, press the ‘Edit’ button next to the ‘Label’ field. This opens a panel in which you can manually enter the labels. Press the green ‘Apply’ button when you have finished entering the labels.



To enter a literal **comma** in the label, escape the comma with a backslash (e.g., “Q1\,25% below, Q2\, 50% below, Q3\,75% below”).

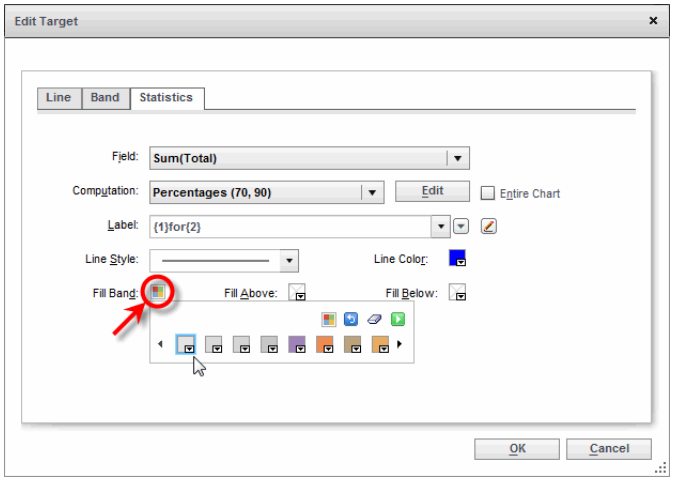
Labels for individual lines should be separated by commas. For example, if you are generating the 4-quantile (which creates three lines), enter three labels separated by commas, e.g., “`Q1: 25% below, Q2: 50% below, Q3: 75% below`”.

If desired, you can add the target value, target formula, and field name into a label by inserting the corresponding codes (`{0},{1},{2}`) shown at the bottom of the panel. You can format the inserted values using the same syntax as *Custom Tooltip*. Some examples are shown below:

```
{1} = {0,number,$#,##0}    yeilds    “70% of Max = $383,485”
{1} for monthly {2}        yeilds    “70% of Max for monthly
Sum(Total)”
```

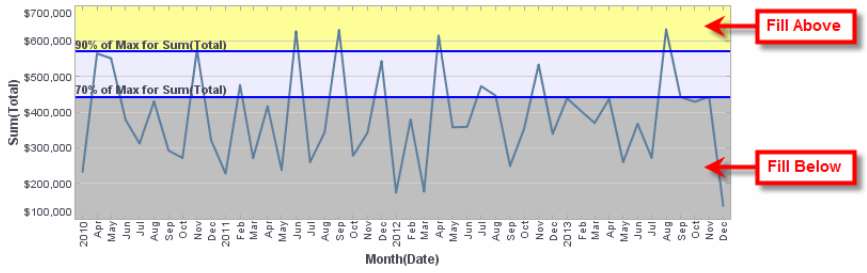
11. Select a ‘Line Style’ and ‘Line Color’ in which to display the statistical lines.

12. (Optional) Press the ‘Fill Band’ button to open a color picker and select a set of colors for the specified bands (i.e., the regions between the statistical lines). Select one color for each band. The colors are applied to the bands from left to right; the left-most color is applied to the lowest band, and so on. When you have selected the desired colors, press the green ‘Apply’ button.



For more information about the features of the color-picker, see *Adding Color Coding to Dimensions*.

13. (Optional) Select a 'Fill Above' color and 'Fill Below' color to fill the regions of the chart above and below the maximum and minimum statistical lines, respectively.



14. Press 'OK' to close the dialog box.

By default, a statistical line appears on the chart even if its value is greater than the largest data point. This may sometimes cause the data points on the chart to be compressed into a small region of the plot area, which makes the chart difficult to read. To correct this, turn off the 'Keep Element in Plot' option in the 'Plot Properties' dialog box. See *Editing Plot Properties* for more information.

See Also

Adding a Target Line, to place a target line or representative line on the chart.

Adding a Target Band, to demarcate a specified data region on the chart.

Adding Property Script (Expressions), in *Advanced Dashboard Scripting*, to set target line values and labels through expressions.

Dynamic Properties, set target line values and labels through *Input Components*.

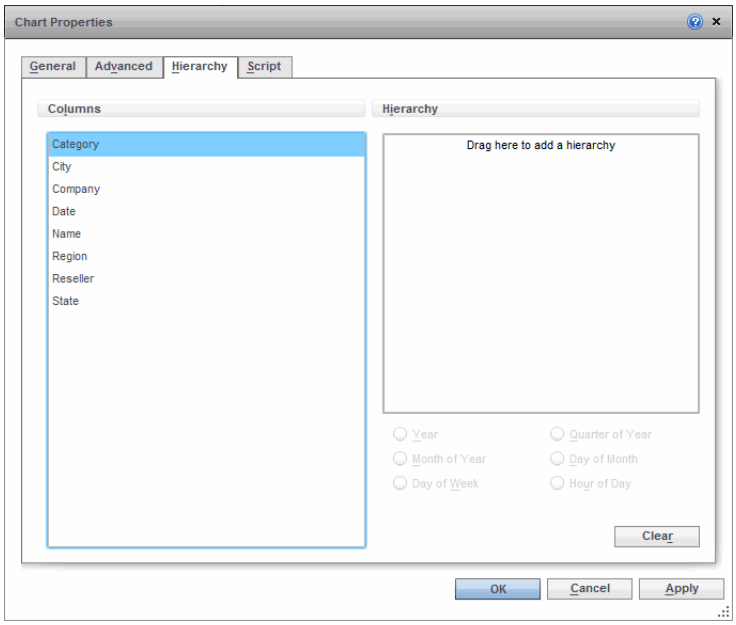
Chart Properties: Hierarchy Tab

The **Hierarchy** tab in the 'Chart Properties' dialog box allows you to specify the hierarchy for a custom drill-down.

To create a custom drill-down hierarchy, follow the steps below:

1. Right-click the Chart, and select 'Properties' from the context menu. This opens the 'Chart Properties' dialog box.

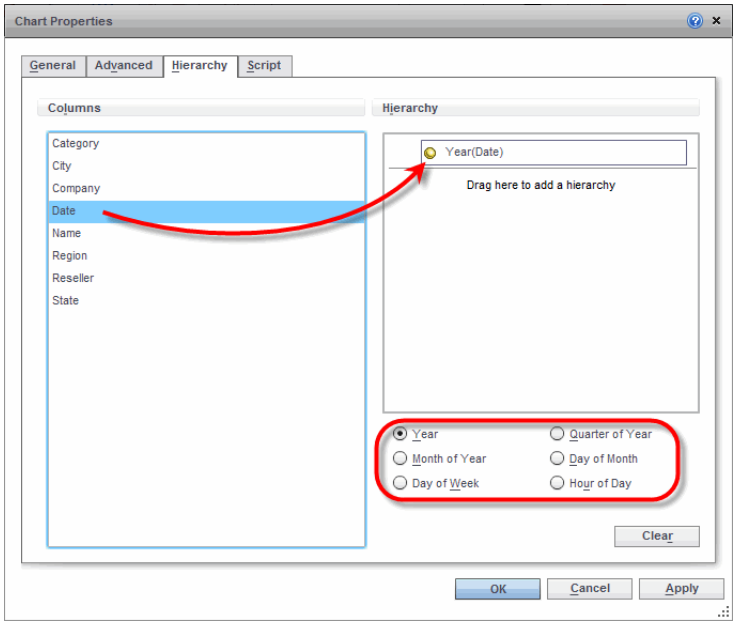
2. Select the **Hierarchy** tab. The 'Columns' panel displays the fields available in the Chart's binding.



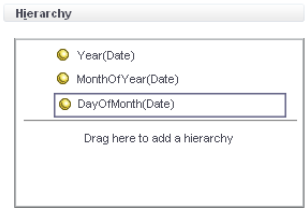
3. Click on a field that you want to include in the hierarchy, and drag the field to the 'Hierarchy' panel.

Drilling on date fields is enabled by default. Use the **Hierarchy** tab to customize the drill sequence.

4. (Optional) If the added field is a date field, select the desired date representation from options below the 'Hierarchy' panel. The options are 'Year', 'Month of Year', 'Day of Week', 'Quarter of Year', 'Day of Month', and 'Hour of Day'.

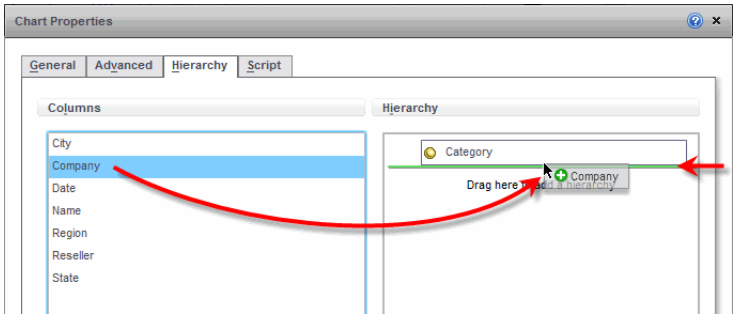


Observe that after you add a date field to the 'Hierarchy' panel, the date field still remains available in the 'Columns' panel. This allows you to add the same date field to the hierarchy at multiple levels, giving each level a different date representation. In this way, you can control the date-drilling sequence. For example, the following hierarchy produces the drill-down sequence 'Year' > 'Month' > 'Day'.



A horizontal green line indicates the position where the field will be placed within the hierarchy

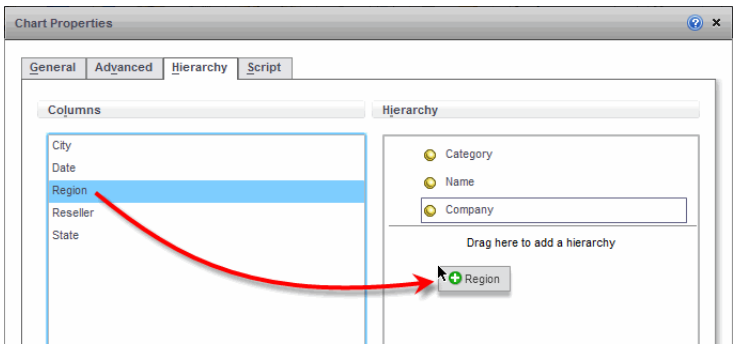
5. Click on another field that you want to include in the hierarchy, and drag it over the existing fields in the 'Hierarchy' panel. Drop the field into the desired position in the hierarchy.



6. Repeat the previous step to add additional fields to the hierarchy, as desired.

   To remove a specific field from the hierarchy, simply drag the field from the ‘Hierarchy’ panel back to the ‘Columns’ panel. To remove *all* fields from the hierarchy, press the ‘Clear’ button.

7. (Optional) To create an additional independent hierarchy, drag a field from the ‘Columns’ panel to the region labeled ‘Drag here to add a hierarchy’. Then repeat the previous steps to add additional fields to this independent hierarchy.



8. Press ‘OK’ to close the dialog box.

Note that there may not be any immediate change to the Chart. However, by configuring a hierarchy as described above, you achieve the following effect: Whenever you (or the end user) adds a field listed on the hierarchy to the ‘X’ or ‘Y’ region of the Chart Editor, the Chart will automatically implement an appropriate drill-down feature based on the specified hierarchy.

For example, consider the following hierarchy:

```
Category
Name
Company
```

After you have specified this hierarchy in the ‘Hierarchy’ panel, if you then bind the ‘Category’ field to the ‘X’ or ‘Y’ region in the Chart Editor, the Chart automatically makes a full drill-down hierarchy (‘Category’ > ‘Name’ > ‘Company’) available to users. Likewise, if you bind the ‘Name’ field to the ‘X’ or ‘Y’ region in the Chart Editor, the Chart automatically makes the appropriate partial drill-down hierarchy (‘Name’ > ‘Company’) available to users.

For information on how to navigate a chart with drill-downs, see *Drilling Down into a Chart*.

See Also

Drilling Down into a Chart, for information on using default drill-down features.

3.4.9 Representing Data with Visual Formats

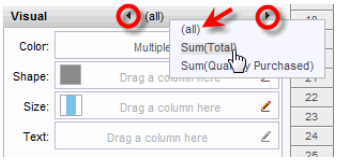
You can represent data values using the following visual formats: Color, pattern, size, or text. These formats can be applied to an entire dataset or to a subseries within a dataset.

The following sections explain how to add visual formats to a dataset. For information on adding formats to subseries, see *Adding a Sub-Series*.

Adding a Fixed Visual Format

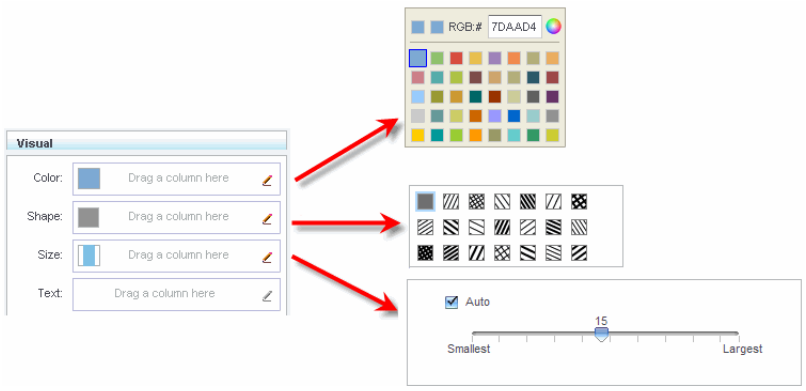
To add a fixed visual format (not keyed to data), follow the steps below:

1. If the chart contains multiple measures with individual styles, click on one of the arrows in the 'Visual' panel title bar to select the particular measure that you want to format. (Or click the area between the arrows to select the measure from a menu.)



   If desired, select 'all' to apply visual formatting (selected in the next step) to all measures.

2. Press the 'Edit' button in one of the format fields in the 'Visual' pane: 'Color', 'Shape', 'Size'.

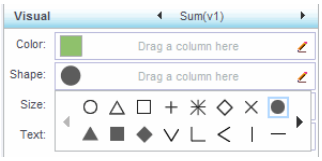


Note: The available shape options vary depending on the Chart type.

The following shapes are available for point-type charts:



To hide a measure's data points in cases where the measure is being used to anchor text on the chart, press the 'Clear' button on the 'Shape' panel. (The 'Clear' button is only available when the 'Text' field is in use.) See *Representing Data With Text* for an example.



An administrator can make custom shapes available by placing image files (png, gif, jpg) in the *WEB-INF/classes/portal/shapes* directory on the server. The icons are sorted alphabetically by file name. Custom shape images should be 18x18 pixels, and will be scaled according to the 'Size' setting in the Chart Editor.

See Also

Setting a Chart Style for an Individual Dataset, to assign individual styles to multiple measures.

Adding Formats to Dimensions, for information on keying formats to grouping.

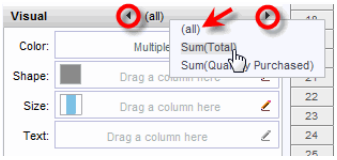
Adding Formats to Measures, for information on keying formats to data values.

Adding a Sub-Series, for information on using format to create groupings.

Adding Formats to Dimensions

To add a visual format to a dimension, follow the steps below:

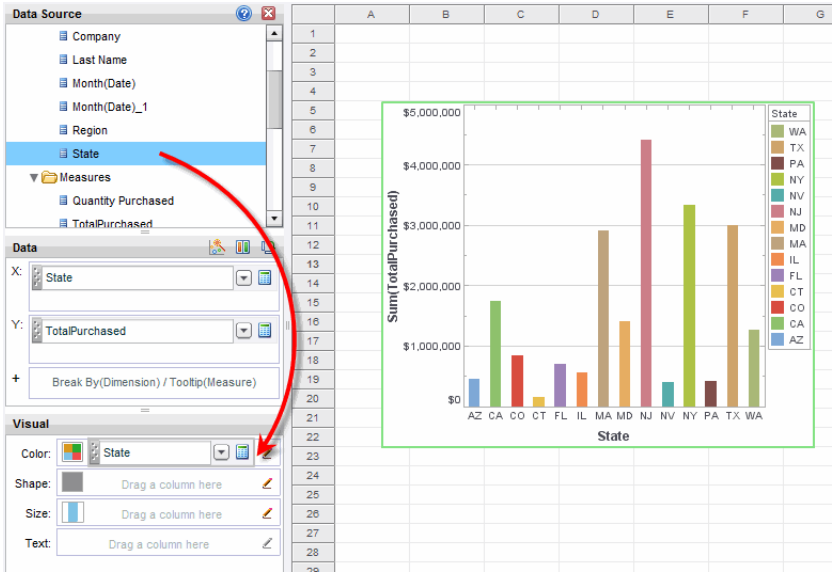
1. If the chart contains multiple measures with individual styles, click on one of the arrows in the 'Visual' panel title bar to select the particular measure that you want to format. (Or click the area between the arrows to select the measure from a menu.)



   If desired, select 'all' to apply visual formatting (selected in the next step) to all measures.

2. Drag the desired dimension from the ‘Data Source’ tree to the desired format field in the ‘Visual’ pane (‘Color’, ‘Shape’, ‘Size’, or ‘Text’).

If the dimension is not already used on the chart X-axis, this creates a new subseries using the specified formatting for the selected measure.



See Also

Converting between Measure and Dimension, to convert a dimension to a measure.

Setting a Chart Style for an Individual Dataset, to assign individual styles to multiple measures.

Positioning a Chart Legend, for information on controlling the legend display.

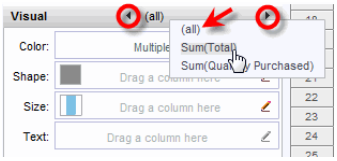
Editing a Chart Legend, for information on changing the display of a legend.

Adding a Sub-Series, for information on using format to create groupings.

Adding Formats to Measures

To add a visual format to a measure, follow the steps below:

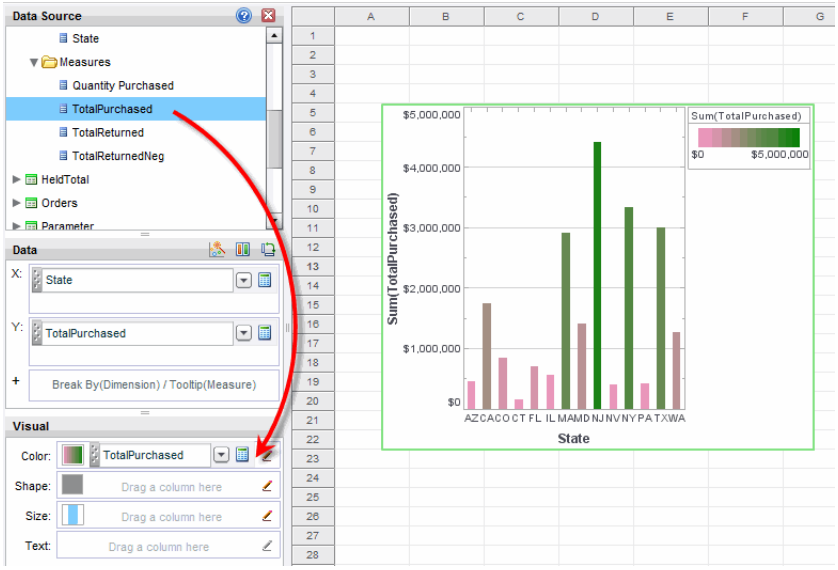
1. If the chart contains multiple measures with individual styles, click on one of the arrows in the ‘Visual’ panel title bar to select the particular measure that you want to format. (Or click the area between the arrows to select the measure from a menu.)



If desired, select ‘all’ to apply visual formatting (selected in the next step) to all measures.

2. Drag the measure (i.e., the measure that will supply the formatting values) from the ‘Data Source’ tree to the desired format field in the ‘Visual’ pane (‘Color’, ‘Shape’, ‘Size’, or ‘Text’).

By default, string-type measures are represented categorically. Other measures are represented linearly unless you select the ‘Discrete’ option. See *Displaying a Measure as Discrete Values* for more information.



See Also

Converting between Measure and Dimension, to convert a measure to a dimension.

Setting a Chart Style for an Individual Dataset, to assign individual styles to multiple measures.

Adding Color Coding to Measures, for information on changing the color representation.

Adding Shape Coding to Measures, for information on changing the shape representation.

Representing Data With Color

To represent data using color, drag a dimension or measure from the **Data Source** panel to the ‘Color’ field in the **Visual** panel.

See Also

Adding Formats to Dimensions, for information on using format to categorize.

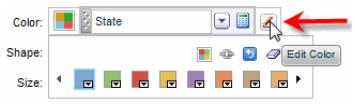
Adding Formats to Measures, for information on using format to code data.

Adding a Sub-Series, for information on using format to create groupings.

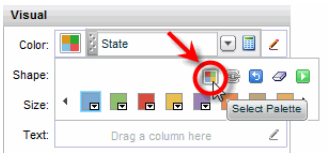
Adding Color Coding to Dimensions

To specify the order in which the colors are applied to the levels of a dimension, follow the steps below:

1. Press the 'Edit' button next to the 'Color' field. This opens the color panel.

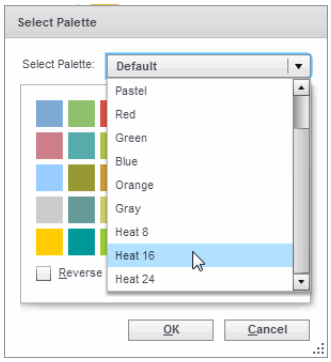


2. (Optional) Press the 'Select Palette' button to open the 'Select Palette' dialog box.



If the selected palette does not contain enough colors to represent all of the distinct data groups, additional groups will be represented by colors from the default palette.

3. Choose the desired palette from the 'Select Palette' menu. If desired, select the 'Reverse' option to reverse the order of the palette colors.

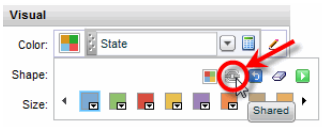


Press 'OK'.

4. Click on the color chips to create the desired color order.

5. Click 'Apply' to finalize the setting.

To reset the color order, click the ‘Reset’ button on the color menu. To synchronize the color mapping to that of other charts (in the Viewsheet) which display the same dimension, click the ‘Shared’ button in the color panel.



To specify one or more custom palettes using your own color choices, create a file called *colorpalettes.xml* in the *portal* directory of SREE Home (*WEB-INF\classes\portal*, by default). The structure of the file should be as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<palettes>
  <palette name="MyPalette_1">
    <colors>
      <color value="#ff0000"/>
      <color value="#00ff00"/>
      <color value="#0000ff"/>
          ...
      <color value="#00ffff"/>
    </colors>
  </palette>
  <palette name="MyPalette_2">
    <colors>
      <color value="#000000"/>
      <color value="#222222"/>
      <color value="#444444"/>
          ...
      <color value="#eeeeee"/>
    </colors>
  </palette>
</palettes>
```

If you designate a custom palette which has the same “`palette name`” as one of the built-in palettes, the custom palette will be used in place of the built-in palette.

See Also

Adding Formats to Dimensions, for information on using format to categorize.

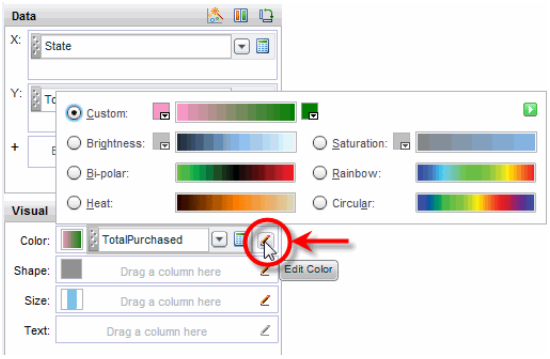
Adding Formats to Measures, for information on using format to code data.

Adding a Sub-Series, for information on using format to create groupings.

Adding Color Coding to Measures

To specify the manner in which colors are used to code the measure values, follow the steps below:

1. Click the ‘Edit’ button next to the ‘Color’ field. This opens the color selection menu.



2. Select a gradient style for the color coding. If required, click the color chip to specify the desired color.

3. Click ‘Apply’ to finalize the setting.

See Also

Adding Formats to Dimensions, for information on using format to categorize.
Adding Formats to Measures, for information on using format to code data.
Adding a Sub-Series, for information on using format to create groupings.

Representing Data With Shape

To represent data using fill pattern or shape, drag a dimension or measure from the **Data Source** panel to the ‘Shape’ field in the **Visual** panel.

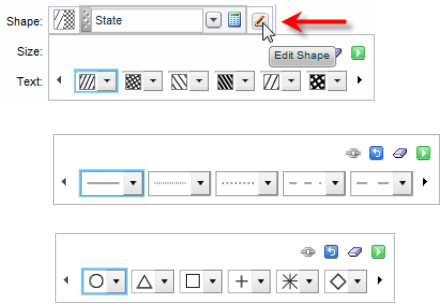
See Also

Adding Formats to Dimensions, for information on using format to categorize.
Adding Formats to Measures, for information on using format to code data.
Adding a Sub-Series, for information on using format to create groupings.

Adding Shape Coding to Dimensions

To specify the order in which the patterns/shapes are applied to the levels of a dimension, follow the steps below:

1. Click the ‘Edit’ button next to the ‘Shape’ field. This opens the shape selection menu.



Note: The available shape options vary depending on the Chart type.

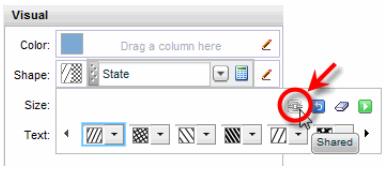
2. Click on the individual shape menus to create the desired pattern/ shape order. The following shapes are available for point-type charts:



An administrator can make custom shapes available by placing the image files (png, gif, jpg) in the *WEB-INF/classes/portal/shapes* directory on the server. The icons are sorted alphabetically by file name. Custom shape images can be any size, and are scaled according to the 'Size' setting in the Chart Editor.

3. Click 'Apply' to finalize the setting.

To reset the shape order, click the 'Reset' button on the shape menu. To synchronize the shape mapping to that of other charts (in the Viewsheet) which display the same dimension, click the 'Shared' button in the shape panel.



See Also

Adding Formats to Dimensions, for information on using format to categorize.

Adding Formats to Measures, for information on using format to code data.

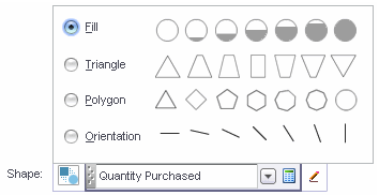
Adding a Sub-Series, for information on using format to create groupings.

Adding Shape Coding to Measures

To specify the manner in which shapes are used to code the measure values, follow the steps below:

1. Click the ‘Edit’ button next to the ‘Shape’ field. This opens the shape selection menu.





Note: The available shape options vary depending on the Chart type.

2. Select a pattern style for the value coding.

   For the ‘Left Tilt’, ‘Right Tilt’ and ‘Grid’ options, the density of the pattern represents the measure value. For the ‘Orientation’ option, the angle of the hashing represents the measure value.

3. Click ‘Apply’ to finalize the setting.

See Also

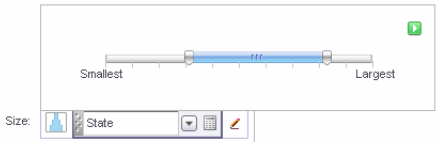
Adding Formats to Dimensions, for information on using format to categorize.
Adding Formats to Measures, for information on using format to code data.
Adding a Sub-Series, for information on using format to create groupings.

Representing Data with Size

To represent data using size, drag a dimension or measure from the **Data Source** panel to the ‘Size’ field in the **Visual** panel.

To specify the range of sizes that should be used in representing the levels of a dimension or values of a measure, follow the steps below:

1. Click the ‘Edit’ button next to the ‘Size’ field. This opens a size range slider.



2. Adjust the slider to select the smallest and largest element size.

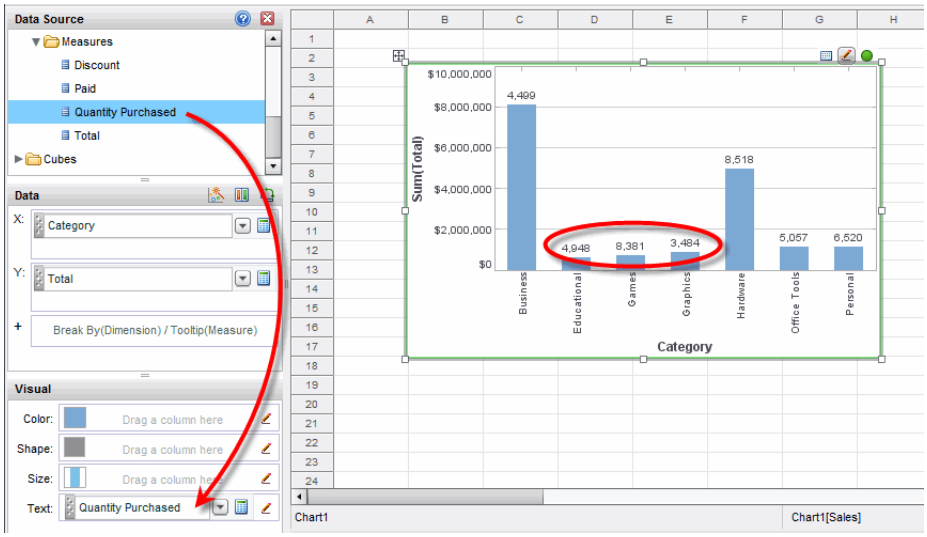
3. Click ‘Apply’ to finalize the setting.

See Also

Adding Formats to Dimensions, for information on using format to categorize.
Adding Formats to Measures, for information on using format to code data.
Adding a Sub-Series, for information on using format to create groupings.

Representing Data With Text

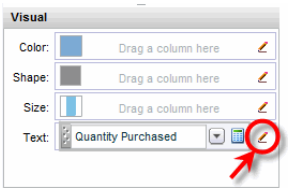
To display a dimension or measure values using text, drag the dimension or measure from the **Data Source** panel to the 'Text' field in the **Visual** panel.



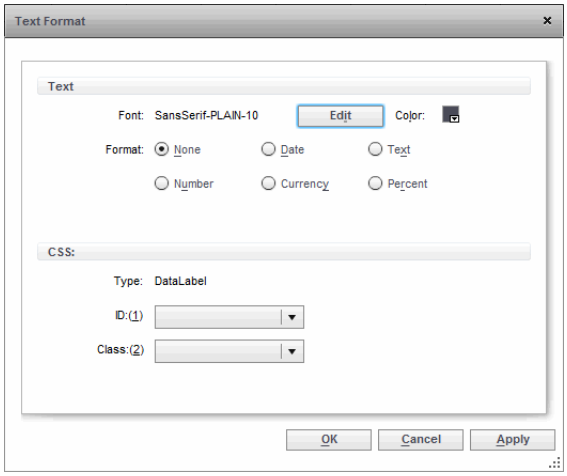
Note: By default, dimension and measure values are automatically displayed by tooltip when the Viewsheet is previewed or deployed.

To format the text displayed for the dimension or measure, follow the steps below:

1. Click the 'Edit' button next to the 'Text' field.



This opens the 'Text Format' dialog box.



2. Select the text size, font, and color.

3. Select the format for the displayed text. See *Format Tab* for an explanation of the format options.

4. Press 'OK' to close the dialog box.

Alternatively, you can click the text on the chart to select it, and then press the desired toolbar buttons to change the formatting.

See Also

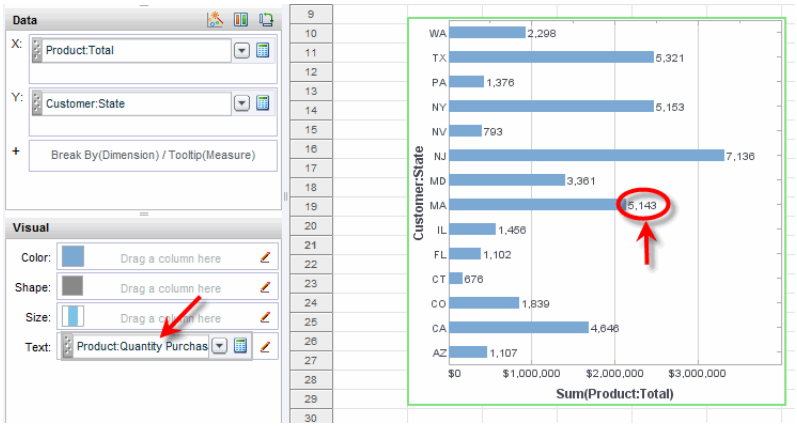
Adding Formats to Dimensions, for information on using format to categorize.

Adding Formats to Measures, for information on using format to code data.

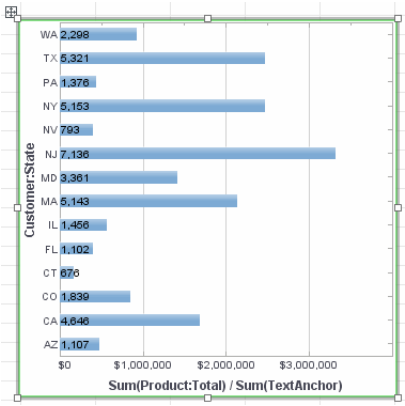
Adding a Sub-Series, for information on using format to create groupings.

Example: Superimposing Measure Values Over a Bar Chart

When you display a measure on a chart as text, the text is placed next to the corresponding data points. For example, in the chart below (based on the 'Order Model' data model), the bars represent the 'Total' values, and the 'Text' binding displays the 'Quantity Purchased' values as text next to the bars.



In some cases, it may be desirable to display the 'Quantity Purchased' value superimposed on top of the bars:



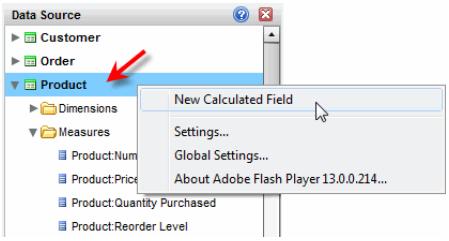
To do this, you can introduce a dummy measure to act as an anchor for the text. Follow the steps below to create this example:

1. Create a new Viewsheet base on the 'Order Model' data model, and drag a Chart component from the **Components** panel into the Viewsheet grid.

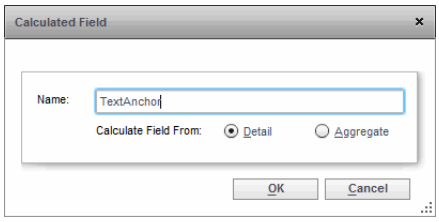
2. From the **Data Source** panel, drag the 'State' field from the 'Customer' entity to the 'Y' region of the **Data** panel.

3. From the **Data Source** panel, drag the 'Total' field from the 'Product' entity to the 'X' region of the **Data** panel.

4. In the **Data Source** panel, right-click on the 'Product' entity, and select 'New Calculated Field'.



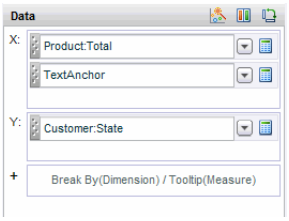
5. Name the field 'TextAnchor' and select the 'Detail' option. Press 'OK'. This opens the Formula Editor.



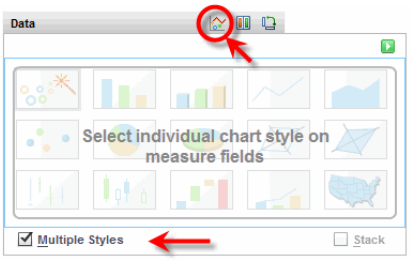
6. In the Formula Editor, set the ‘Return Data Type’ to ‘Double’, enter the formula “1”, and press ‘OK’.

   This creates a new measure that has the uniform value “1”. You will use this measure to anchor the ‘Quantity Purchased’ values to the left side of the plot region.

7. Drag the new ‘TextAnchor’ measure from the **Data Source** panel to the ‘X’ region of the **Data** panel, under the ‘Total’ field.



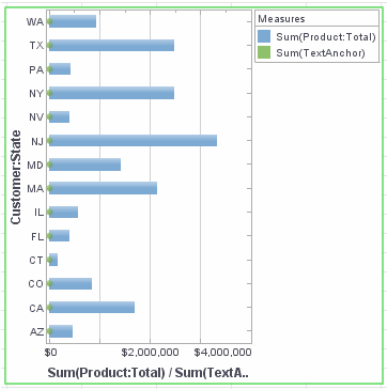
8. Press the ‘Select Chart Style’ button, and enable the ‘Multiple Styles’ option. Press the green ‘Apply’ button to close the panel.



9. Press the ‘Select Chart Style’ button next to the ‘Total’ field, and switch the chart style to ‘Bar’.

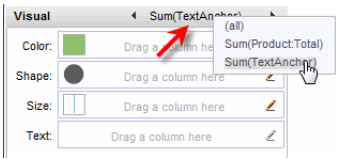
10. Press the ‘Select Chart Style’ button next to the ‘TextAnchor’ field, and switch the chart style to ‘Point’.

11. Press the ‘Switch to Single Graph’ button to place both data sets on the same axis.

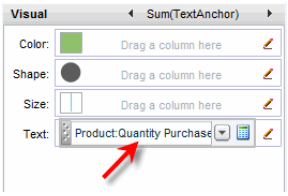


12. Right-click on the legend, and select ‘Hide Legend’.

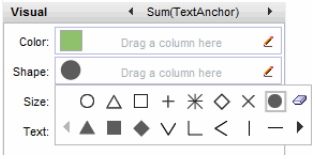
13. In the **Visual** panel, click the title bar between the arrows and select the ‘Sum(TextAnchor)’ measure.



14. From the **Data Source** panel, Drag the ‘Quantity Purchased’ field from the ‘Product’ entity to the ‘Text’ region of the **Visual** panel.



15. In the **Visual** panel, press the ‘Edit’ button in the ‘Shape’ region, and press the ‘Clear’ button. This hides the points for the ‘TextAnchor’ measure.

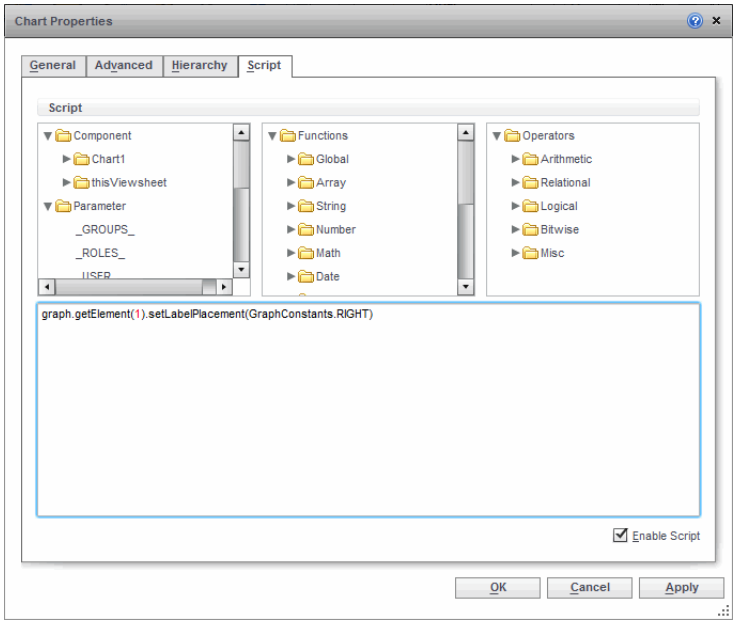


16. In the **Visual** panel, press the ‘Edit’ button in the ‘Color’ region, and set a black color for the text.

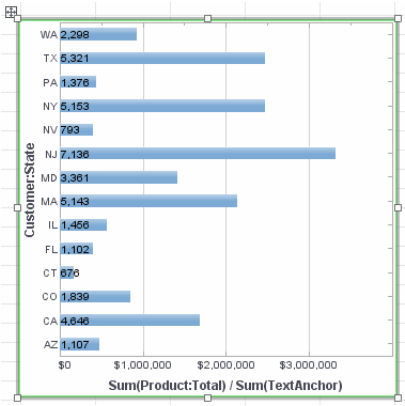
17. Right-click on the chart and select ‘Properties’. This opens the ‘Chart Properties’ dialog box.

18. Select the **Script** tab, and enter the following script, and press ‘OK’.

```
graph.getElement(1).setLabelPlacement(GraphConstants.RIGHT)
```



This sets the alignment appropriately, and yields the desired result:

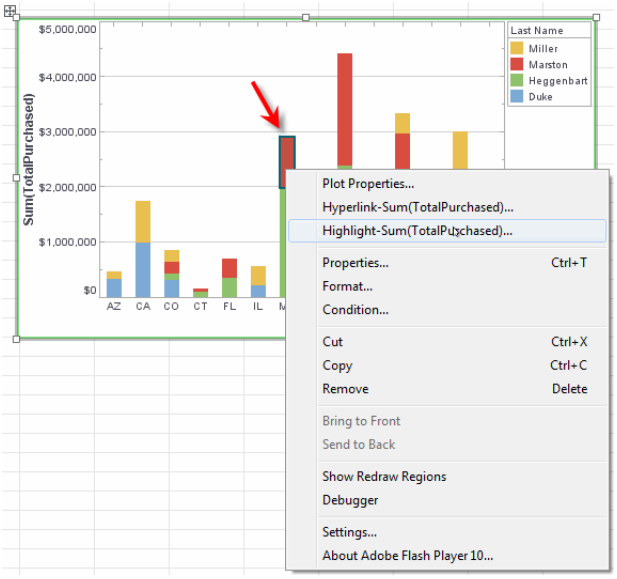


¢

3.4.10 Adding Highlights to a Chart

You can highlight chart elements based on the values that they contain. To add highlights to a chart, follow these steps:

1. Right-click on a data group in the chart, and select ‘Highlight’ from the context menu. This opens the ‘Highlight’ dialog box.



2. In the 'Highlight' dialog box, define the desired highlight. See the *Highlights* section for details on how to do this.

Example: Chart Highlighting

In this example, you will create a chart to display 'Quantity Purchased' vs. 'State', and use highlighting to draw attention to values that exceed 5000 or fall below 1000. Follow the steps below:

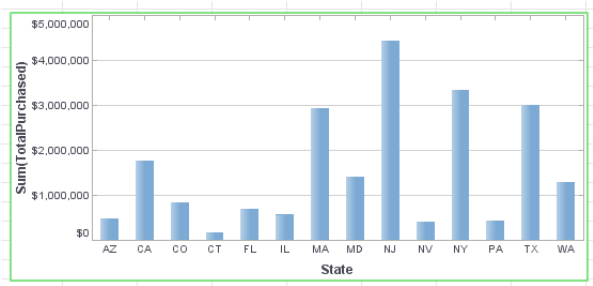
1. Create a new Viewsheet based on the sample 'Analysis' Worksheet.

2. Add a chart component, and click the 'Edit' button. This opens the Chart Editor.

3. In the **Data Source** panel, expand the 'OrdersAndReturns' data block.

4. From the 'Dimensions' node, drag the 'State' field to the 'X' field of the **Data** panel.

5. From the 'Measures' node, drag the 'Quantity Purchased' field to the 'Y' field of the **Data** panel. This creates the desired chart.



6. Right-click in any bar on the chart, and select the ‘Highlight-Quantity Purchased’ option from the context menu. This opens the ‘Highlight’ dialog box.

7. Click the ‘Add’ button to open the ‘Name’ dialog box.

8. Enter ‘Under1000’ as the name of the first highlight. Click ‘OK’ to close the dialog box. The new highlight name appears in the highlight list.

9. Click the highlight name in the list to select it. From the ‘Color’ palette, select the color red (rgb value “FF0000”).

10. In the ‘Condition’ panel, click the ‘Edit’ button to open the ‘Condition’ dialog box. Enter the following condition, and click ‘OK’.

```
[Sum of Quantity Purchased][is][less than][1000]
```

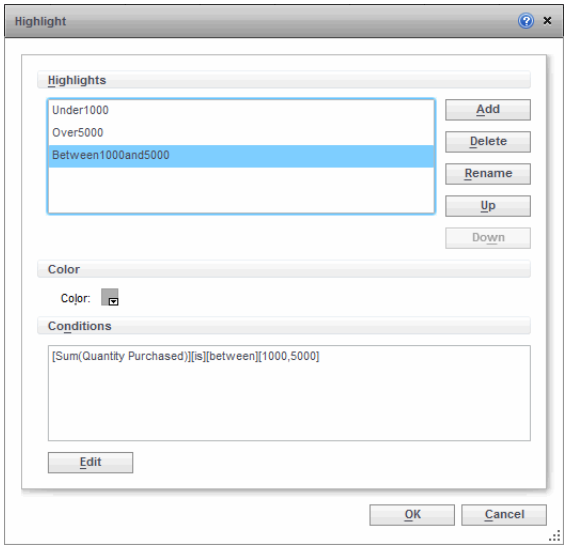
11. Repeat Steps 7-10 to add the following conditions and formats:

    a. Name = ‘Over5000’, rgb value = ‘AAFFAA’

    ```
    [Sum of Quantity Purchased][is][greater than][5000]
    ```

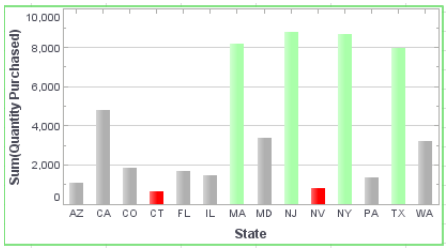
    b. Name = ‘Between1000and5000’, rgb value = ‘B0B0B0’

    ```
    [Sum of Quantity Purchased][is][between][1000,5000]
    ```



12. Click ‘OK’ to exit the ‘Highlight’ dialog box.

The chart table now shows the value-keyed highlighting you specified. In particular, the very large and very small order quantities have been highlighted in green and red, respectively.



¢

See Also

Highlights, for information on defining a highlight.

Brushing a Chart, for information on highlighting data across multiple charts.

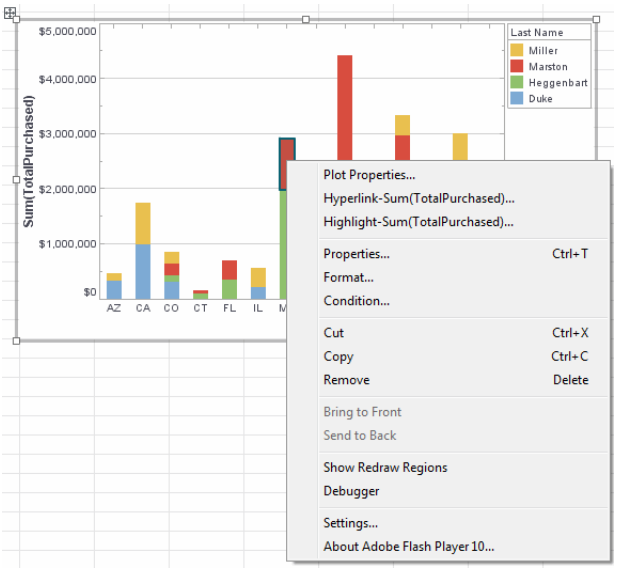
Crosstabs, for information on creating crosstabs.

3.4.11 Adding Hyperlinks to a Chart

In charts, hyperlinks can be attached to the chart graphical elements (e.g., the bars in a bar chart). To create hyperlinks on a chart, follow the steps below:

1. Right-click on a chart element to add hyperlinks to the chart elements (e.g., bars), or right-click on a chart dimension label to add hyperlinks to the labels.

2. Select ‘Hyperlink’ from the context menu. This opens the ‘Hyperlink’ dialog box.



3. In the ‘Hyperlink’ dialog box, define the desired hyperlink. See the *Hyperlinks* section for details on how to do this.

See Also

Hyperlinks, for information on the ‘Hyperlink’ dialog box.

3.5 Map Charts

A map is a type of chart, and displays summarized data grouped by physical location in the form of a geographical map. It has many of the same properties of other chart types, but also differs in some ways. The following sections explain how to create and modify map charts.

See Also

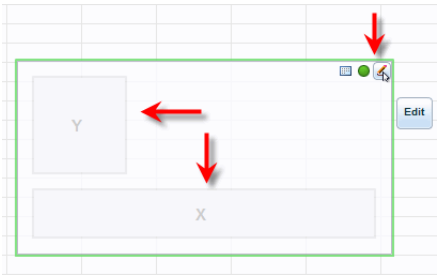
Creating a Custom Map, in *Advanced Server Administration*, to create a map based on a custom shapefile.

3.5.1 Creating a Map Chart

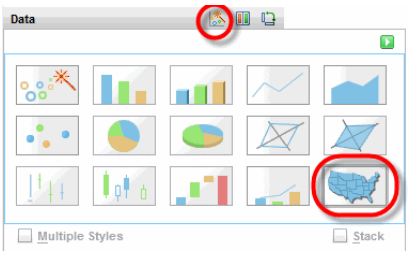
To add a map chart to a Viewsheet, drag and drop the Chart component from the Component tree into the grid. Use the mouse to move and resize the chart as desired.

To bind the Chart to geographical data, follow the steps below:

1. Click on the center of the chart, or click the ‘Edit’ button in the top-right corner. This opens the Chart Editor on the left side of the Visual Composer.



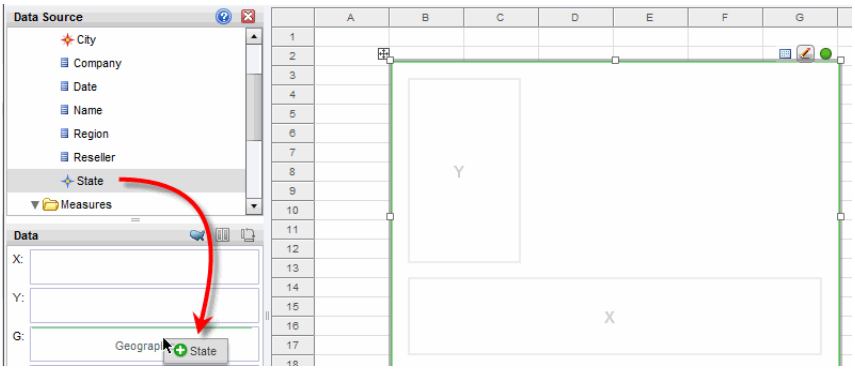
2. In the Chart Editor's **Data** panel, click 'Select Chart Style', and choose the 'Map' chart type. Click the 'Apply' button (or double-click the 'Map' icon).



The the **Data** panel changes to display a 'Geographic' field.

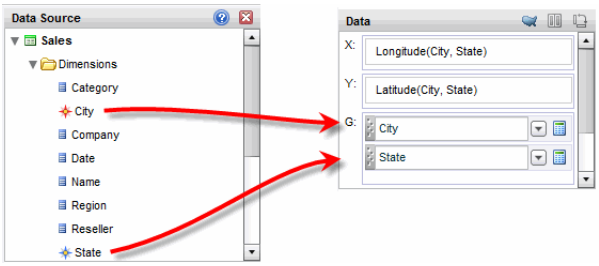
If your columns contain longitude and latitude data (in decimal format) rather than location names, use the 'Longitude' and 'Latitude' fields of the 'Data' panel.

3. From the **Data Source** panel in the Chart Editor, drag a geographic dimension onto the 'Geographic' field in the **Data** panel. (See *Setting a Geographical Field* to declare a field as geographical.)

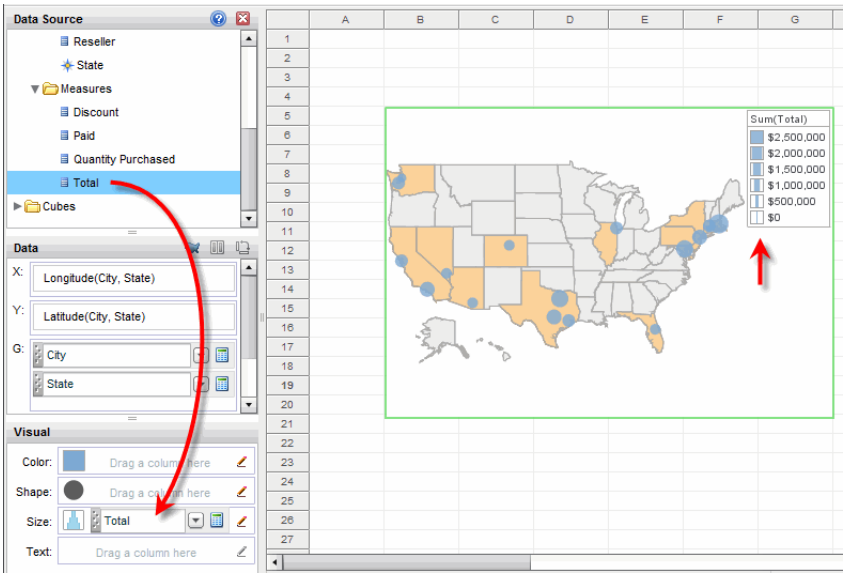


The chart updates to display the locations contained in the geographic field.

4. Repeat the previous step to bind additional 'Geographic' levels. For example, you can bind 'City' and 'State' fields.

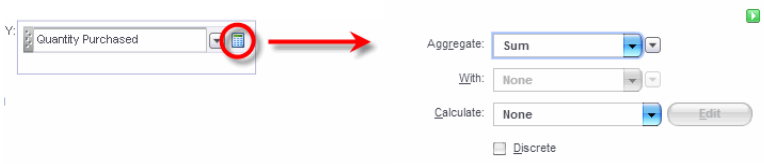


5. From the **Data Source** panel, drag a measure onto one of the fields in the **Visual** panel ('Color', 'Shape', 'Size', or 'Text').

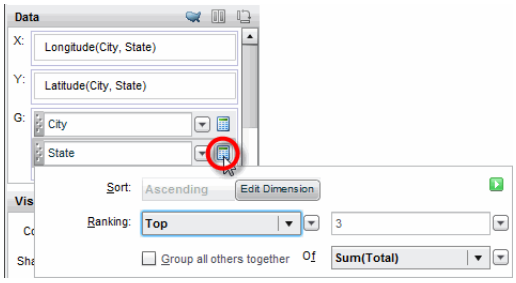


This binding will display the measure on the chart using the particular visual coding you selected, and automatically creates a corresponding legend. See *Adding a Sub-Series* for more information on using the **Visual** panel.

6. Click the 'Edit Measure' button, and select the desired aggregation method for the measure.

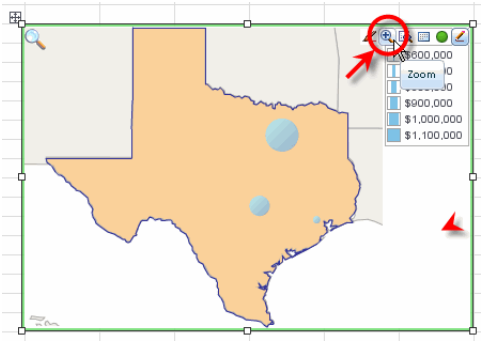


7. (Optional) For any geographical fields that you specified, click the 'Edit Dimension' button, and select the desired ranking for the geographical data. For example, you can display just the top three states according to the measure.



8. (Optional) To create a particular trajectory through the points on the chart, drag a *path* field to the 'P' region. The values of this field are used to assign the sort order for the plotted data so that connecting lines are drawn between points which are adjacent in the *path* sort order.

9. (Optional) Set the initial map view by selecting a region, and clicking the 'Zoom' button. See *Zooming a Chart* for more details.



See Also

Creating a Custom Map, in *Advanced Server Administration*, to create a map based on a custom shapefile.

Setting a Geographical Field, for information on declaring a field geographical.

Editing Chart Formats, for information on changing chart labels.

Adding a Sub-Series, for information on using the **Visual** panel.

Creating Selection List from Chart Data Source, for information on adding a Selection List.

Creating a Selection Tree from Chart Data Source, for information on adding a Selection Tree.

Positioning a Component, for information on moving and resizing a chart.

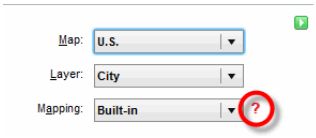
Resolving Geographical Data

When Visual Composer cannot automatically match your geographical data to the locations in the geographical database, you can use the 'Geographic Mapping' dialog box to manually create the correct mapping.

To set the mapping for geographical data, follow the steps below.

1. In the Chart Editor, right-click the desired geographical field, and select 'Edit Geographic' to open the editing panel. (See *Setting a Geographical Field* for more information.)

   The Chart Editor attempts to resolve values in the selected geographic field against the corresponding layer in the map database. If all data values are successfully matched, a green check mark is displayed. If some data values cannot be matched, a red question mark is displayed.

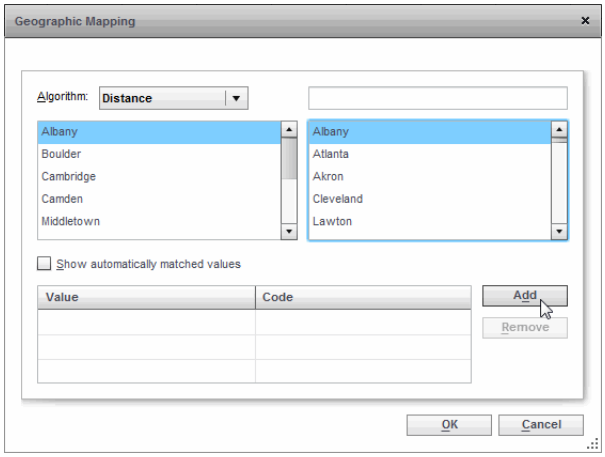


2. To resolve any unmatched names, click the red question mark. This opens the 'Geographical Mapping' dialog box. In the 'Geographical Mapping' dialog box, follow the steps below:

   a. Select one of the unmatched names in the top left panel. The right panel will display a list of candidate matches. (If the list is very long, type a portion of the name in the search box above the panel to limit the results.)

   b. If the list does not contain the intended match, select another matching method from the 'Algorithm' menu.

   c. Select the correct match and click the 'Add' button.



   This creates the desired mapping between the data value and the geographical information in the map database. (You can match multiple data values to a single geographical location.)

3. Repeat the above steps until all values have been matched.

4. When you have completed making all matches, and the left panel is empty, click ‘OK’ to close the dialog box.

5. Click the ‘Apply’ button in the geographical editing panel.

To reuse a custom mapping, create and save the mapping in Style Studio. See *Resolving Geographical Data* in *Advanced Report Design* for more details.

Example: Resolving Geographical Data

In this example, you will create a Worksheet Data Block that contains state names that have an unusual format. Because of the unusual format, you will need to manually map these state name to the correct geographical locations.

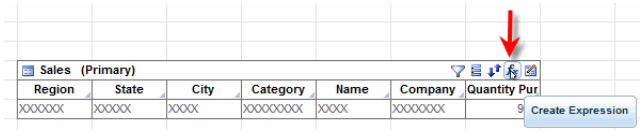
This example assumes that you are familiar with Data Worksheets, and requires some scripting. See *Creating an Expression/Formula Column* in *Advanced Data Mashup* for complete information.

1. Open the ‘US Sales’ Worksheet from the ‘Tutorial Folder’.

2. Save a copy of the Worksheet as ‘US Sales Copy’.

3. In the ‘US Sales Copy’ Worksheet, create a formula column that contains variations of the state names in the ‘State’ column. To do this, follow the steps below:

   a. Click the ‘Create Expression’ button in the title bar of the ‘Sales’ table.



   b. When Visual Composer prompts you for the expression name, enter ‘New State’ for ‘Expression Name’. Click ‘OK’ to open the Formula Editor.

   c. Deselect the ‘SQL’ option. The formula will use JavaScript syntax rather than SQL.

   d. In the Formula Editor, enter an expression that will change two of the names in the ‘State’ column:

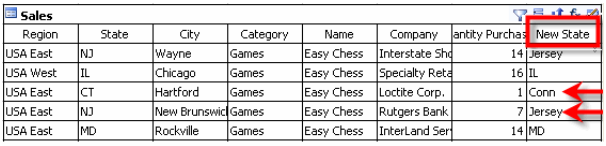
```
if (field['State'] == 'NJ') {
   'Jersey';
}
else if (field['State'] == 'CT') {
   'Conn';
}
else {
```

```
    field['State']
}
```

This simply changes the values ‘NJ’ and ‘CT’ to ‘Jersey’ and ‘Conn’ in the expression column, respectively, leaving all other state values the same.

Note: Click the ‘Show Live Data’ button in the table title bar to view the actual data.

e. Click ‘OK’ to exit the Formula Editor. This creates the new column called ‘New State’.



Sales

Region	State	City	Category	Name	Company	antity Purchas	New State
USA East	NJ	Wayne	Games	Easy Chess	Interstate Sho	14	Jersey
USA West	IL	Chicago	Games	Easy Chess	Specialty Reta	16	IL
USA East	CT	Hartford	Games	Easy Chess	Loctite Corp.	1	Conn
USA East	NJ	New Brunswick	Games	Easy Chess	Rutgers Bank	7	Jersey
USA East	MD	Rockville	Games	Easy Chess	InterLand Ser	14	MD

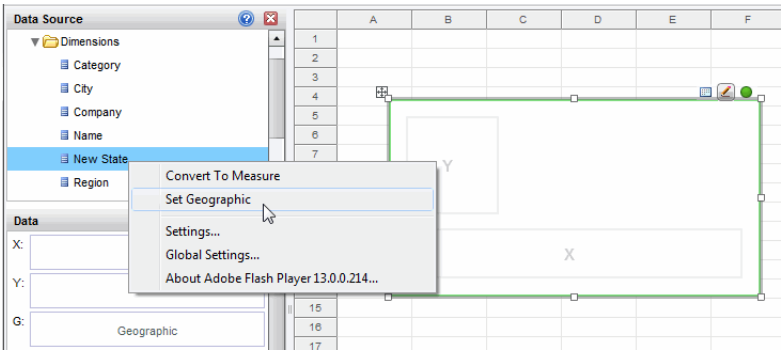
4. Save the Worksheet.

5. Create a new Viewsheet based on this ‘US Sales Copy’ Worksheet.

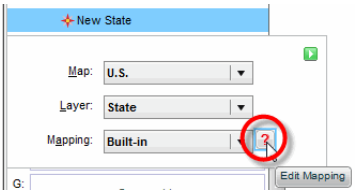
6. Add a Chart to the new Viewsheet, and click the ‘Edit’ button on the Chart to open the Chart Editor.

7. In the **Data** panel of the Chart Editor, click the ‘Select Chart Style’ button, and choose the ‘Map’ type.

8. In the **Data Source** panel of the Chart Editor, right-click the ‘New State’ field, and select ‘Set Geographic’. This opens the geographical editing panel.



9. Click the red question mark to open the ‘Geographical Mapping’ dialog box.



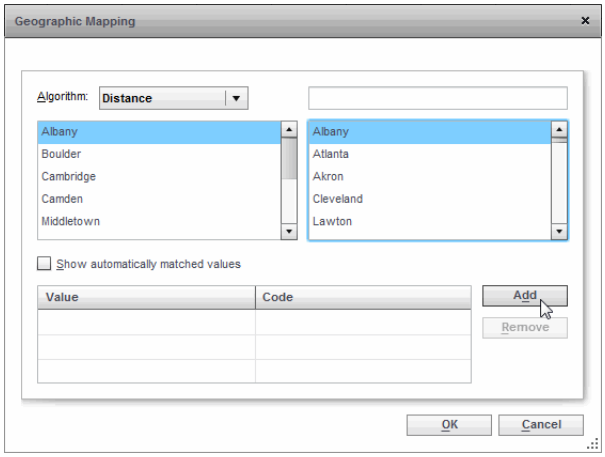
Observe that in the left panel of the ‘Geographical Mapping’ dialog box, two states are not mapped: ‘Jersey’ and ‘Conn’.

10. Resolve these unmatched states manually. Follow the steps below:

If the suggestions are unsatisfactory, select a different matching method from the ‘Algorithm’ menu.

a. In the left panel, select the state ‘Conn’. In the right panel, Visual Composer suggests a set of likely matches.

b. Select the correct match (‘Connecticut’), and click the ‘Add’ button.



This creates a new mapping between the value ‘Conn’ and the location ‘Connecticut’, and removes ‘Conn’ from the left panel.

a. Click ‘Jersey’ in the left panel. In the right panel, Visual Composer suggests a set of likely matches.

You can start typing “Jersey” in the search box to limit the results.

b. Select the correct match (‘New Jersey’), and click the ‘Add’ button. This creates a new mapping between the value ‘Jersey’ and the location ‘New Jersey’, and removes ‘Jersey’ from the left panel.

11. Since all unmatched locations have now been resolved, click ‘OK’ to close the ‘Geographic Feature Names’ dialog box.

12. Click ‘Apply’ in the geographical editing panel for the ‘New State’ field.

13. Drag the ‘New State’ field from the **Data Source** panel to the ‘Geographic’ region of the Chart Editor **Data** panel.

14. Drag the ‘Quantity Purchased’ measure from the **Data Source** panel to one of the **Visual** panel regions (‘Color’, ‘Shape’, etc.) to complete the chart.

¢

Using Latitude and Longitude Data

You can use raw latitude and longitude to designate location, rather than specifying place names (city, state, etc.). The following example provides an illustration.

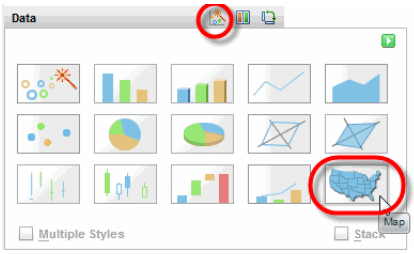
Walkthrough

In this example, you will label a map with a list of landmarks along Historic Route 66. The locations of the landmarks are provided by a Worksheet Data Block containing raw latitude and longitude. Follow the steps below:

1. Create a new Viewsheet based on the ‘Tutorial’ > ‘Map Points’ Worksheet.

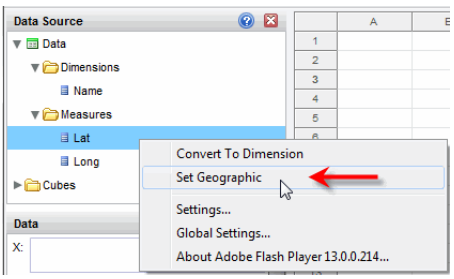
2. Add a Chart component to the Viewsheet, and click the center of the Chart to open the Chart Editor.

3. In the **Data** panel, click the ‘Select Chart Style’ button. Select the ‘Map’ type, and click the green ‘Apply’ button.

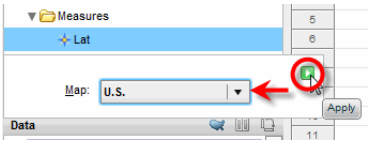


The ‘Lat’ field contains latitude values.

4. In the **Data Source** panel, right-click the ‘Lat’ measure, and select ‘Set Geographic’ from the context menu.



5. From the ‘Map’ menu, select ‘U.S.’, and click the green ‘Apply’ button.

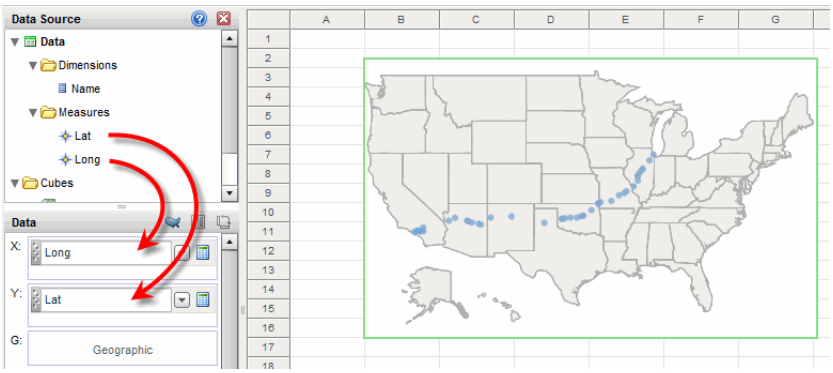


The ‘Long’ field contains longitude values.

6. Repeat the above steps to set the ‘Long’ measure as a geographic field.

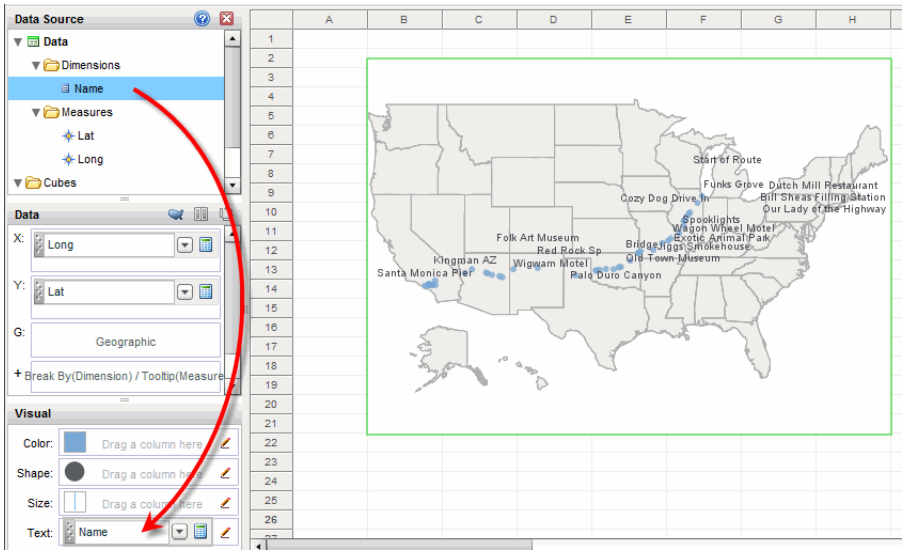
7. Drag the ‘Long’ field to the ‘X’ region of the **Data** panel.

8. Drag the ‘Lat’ field to the ‘Y’ region of the **Data** panel.



Data points now appear on the map to mark the locations specified by the latitude and longitude data, and outline the progression of Historic Route 66.

9. To add the landmark labels to the points on the map, drag the ‘Name’ field from the **Data Source** panel to the ‘Text’ field in the **Visual** panel.



Note: By default, the landmark names are displayed as tooltips.

The labels appear fairly crowded because of the long landmark names. You can zoom the chart to see the labels more clearly. See *Zooming a Chart* for instructions.

Transforming Longitude/Latitude for Alaska and Hawaii

For convenience, the default 'U.S.' map does not display Alaska and Hawaii in their geographical locations, but shifts and (in the case of Alaska) re-scales them to an inset position below California. For this reason, you should transform your raw latitude and longitude data for Alaska and Hawaii prior to generating the chart.

To remap your raw longitude and latitude data, [*long lat*], to the corrected values, [*long' lat'*], apply the following matrix transformations:

Figure 8. Transformation for Alaska

$$\begin{bmatrix} long' \\ lat' \\ 1 \end{bmatrix} = \begin{bmatrix} 0.204 & 0 & -85.0 \\ 0 & 0.354 & 5.0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} long \\ lat \\ 1 \end{bmatrix}$$

Figure 9. Transformation for Hawaii

$$\begin{bmatrix} long' \\ lat' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 50.0 \\ 0 & 1 & 5.0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} long \\ lat \\ 1 \end{bmatrix}$$

3.5.2 Map Chart Properties

The map chart is a type of chart, and provides most of the same properties as other chart types. See *Chart Properties* for more information.

See Also

Properties Dialog Box, for information about common component properties.

4 Data View Features

The following sections present features that facilitate display and analysis of data by Data View components (Charts, Tables, Crosstabs). These features include highlighting, filtering, hyperlinking, and calculation.

4.1 Highlights

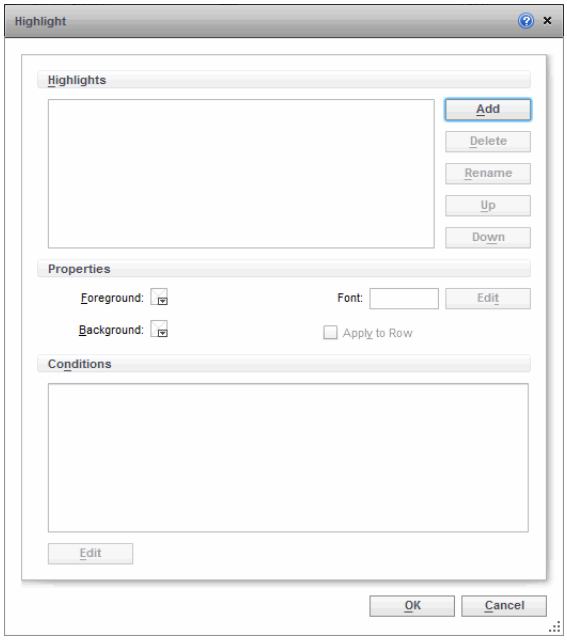
You can apply conditional formatting to table columns, images, text elements, crosstabs, and charts, so that the element formatting changes based on the data bound to the element. These conditional formats are called *highlights*. They are very useful for calling attention to an element when its associated data meets (or fails to meet) some specified condition.

4.1.1 Creating a Highlight

To apply a highlight to a table column, image, text element, chart, or crosstab, follow these steps:

1. Right-click the element you wish to highlight. (For a Table, right-click a data cell in column to which you want the highlight to be applied.)

2. Select ‘Highlight’ from the right-click context menu. This opens the ‘Highlight’ dialog box.



3. Click the ‘Add’ button. This opens the ‘Name’ dialog box.

4. Enter a name for this highlight, and click ‘OK’ to close the dialog box.

5. Repeat the previous steps to enter any additional highlights, and then follow the steps below for each individual highlight.

6. Select the highlight in the list that you want to edit.

7. In the ‘Properties’ panel, select the highlight format. The formatting is applied when the data meets the highlight condition (specified below):

   a. Click the ‘Foreground’ color chip to set the text color.

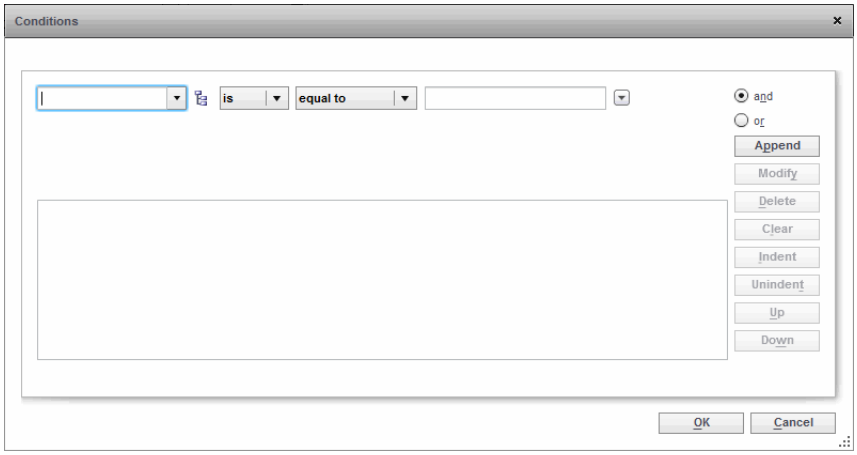
   b. Click the ‘Background’ color chip to set the fill color.

   c. Click the ‘Edit’ button next to the ‘Font’ field to open the ‘Font’ dialog box. Make the desired font selections and press ‘OK’. (Press ‘Reset’ to restore the original settings.)

Cell-based highlights take precedence over row-based highlights.

8. For a Table, select ‘Apply to Row’ if you wish to apply the specified formatting to the entire table row. Otherwise, the formatting applies only to the selected table column.

9. Click the ‘Edit’ button below the ‘Conditions’ box. This opens the ‘Conditions’ dialog box.



Note: To use aggregate fields in a highlight condition for a Freehand Table, first assign a ‘Cell Name’ to the aggregate field in the Freehand Table Editor. See *Freehand Tables* for more information.

10. Follow the steps below to specify the condition expression for this highlight:

a. Select the operand and type of comparison from the menus at the top of the dialog box, e.g., "`[Company][is not][one of]`"

   For table highlights, the operand is one of the table column names. For text and image highlights, the operand is called '`value`', which is the aggregate value associated with the element, as configured in the **Data** tab of the element's 'Properties' dialog box.



b. For table highlighting, select 'Value', 'Field', or 'Variable' from the 'Change Value Type' popup menu (triangle).

c. If you select the 'Field' option, in the adjoining menu, choose the table column for comparison. (The two columns will be compared row by row, and the specified highlight will be applied to every row where the condition holds true.)

d. If you select the 'Value' option, either enter the comparison value into the provided text field, or select 'Browse Data' to choose the comparison value from a list.

e. If you select the 'Variable' option, select one of the *Input Components* listed in the menu to provide the value at runtime. (See *Dynamic Properties* for further details.)

f. If you select the 'Expression' option, press the 'js'/'sql' button to select JavaScript or SQL syntax, and then press the 'Edit' button. This opens the Script Editor where you can enter your expression. (See *Adding Property Script (Expressions)* in *Advanced Dashboard Scripting* for details on using expressions.)

g. When you have fully specified the condition, click 'Append'.

h. To create a compound condition, repeat Steps a-g above, using the 'and' and 'or' operators to composite the conditions. See *Modifying a Filter/Highlight Condition* for more details.

i. Click 'OK' to exit the 'Conditions' dialog box and return to the 'Highlight' dialog box.

When you add a highlight to a table column or crosstab, the highlight condition is tested individually for each cell. When the condition is satisfied, the corresponding highlight format is applied to the cell or row (if 'Apply to Row' is selected).

You can use multiple highlights in combination to produce composite formats. For example, if you specify a red-foreground highlight for the

condition “Discount is greater than 0,” and you also specify a bold font highlight for the condition “State is equal to NJ,” values which meet both conditions will appear highlighted in red *and* bold.

See Also
Adding Highlights to a Table, for information on the table ‘Highlight’ dialog box.
Adding Highlights to Text, for information on the text ‘Highlight’ dialog box.
Adding Highlights to an Image, for information on the image ‘Highlight’ dialog box.
Adding Highlights to a Chart, for information on highlighting charts.
Filtering Operators, in *Advanced Data Mashup*, for explanation of the operators.
Advanced Dashboard Scripting, for more sophisticated conditional formatting.

4.1.2 Modifying a Filter/Highlight Condition

You can make a variety of modifications to conditions in the ‘Conditions’ dialog box. The following sections discuss these options.

See Also
Highlights, for information about highlighting conditions.
Filter Conditions, for information about filtering conditions.

Inserting a Filter/Highlight Condition

To insert a new condition (clause) into an existing condition set, follow these steps:

1. In the ‘Conditions’ dialog box, select an existing condition. The new condition will be inserted above the selected condition. (To reorder the condition after insertion, see *Nesting/Ordering a Filter/Highlight Condition*.)

2. Construct the new condition as described in Step 9 of the *Highlights* section.

3. Select the desired connector, ‘and’ or ‘or’, by which the new conditions should be connected with the previous conditions.

4. Click ‘Insert’.

See Also
Highlights, for information about highlighting conditions.
Filter Conditions, for information about filtering conditions.

Modifying a Filter/Highlight Condition or Connector

To modify an existing condition, follow these steps:

1. In the ‘Conditions’ dialog box, select the existing condition (clause) you wish to modify.
2. Edit the condition as desired using the menus at the top of the dialog box.
3. Click ‘Modify’

To modify the connector between conditions (clauses), follow these steps:

1. In the ‘Conditions’ dialog box, select the connector (‘and’ or ‘or’) you wish to change.
2. Select the desired connecting using the ‘and’ and ‘or’ buttons at the top of the dialog box.
3. Click ‘Modify’.

To delete a condition (clause), select the condition, and click ‘Delete’. To delete all conditions (clauses), click ‘Clear’.

See Also
Highlights, for information about highlighting conditions.
Filter Conditions, for information about filtering conditions.

Nesting/Ordering a Filter/Highlight Condition

To specify that a particular operation (clause evaluation) should be performed before others, follow these steps:

1. Select the connector (‘and’ or ‘or’) that you wish to evaluate first.
2. Click ‘Indent’.

The connector and its accompanying clauses are indented to indicate that they will be evaluated first. Indenting is equivalent to placing parentheses around the specified clauses.

Reordering a Filter/Highlight Condition

To change the position of a clause in the condition, select the desired clause, and click ‘Up’ or ‘Down’.

If the condition contains both ‘and’ and ‘or’ connectors, in general, the reordering of clauses will change the meaning of the condition.

See Also
Highlights, for information about highlighting conditions.
Filter Conditions, for information about filtering conditions.

4.2 Filter Conditions

You can filter the values displayed in a data view component (Chart, Table, Crosstab) or output component (Gauge, Thermometer, Text, etc.) to match specified criteria by setting a *Condition* on the component. For example, you can filter a chart or a table so that it displays only data for a particular sales employee or region.

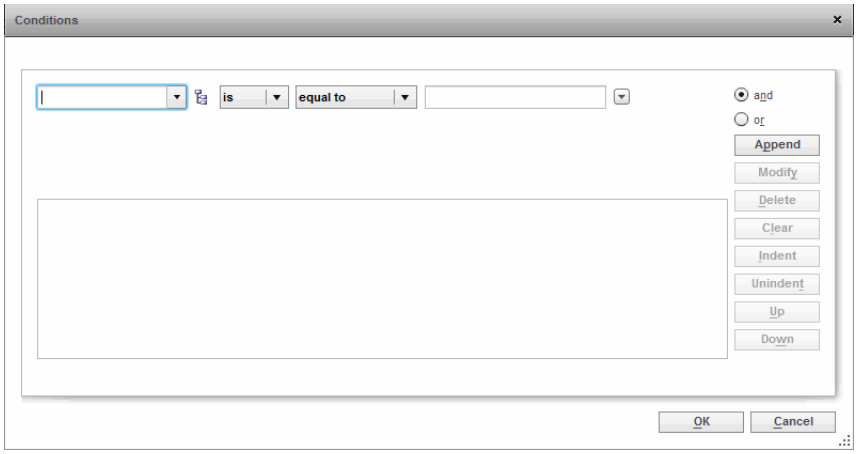
Condition-based filtering applies on the component level, and is different than filtering provided by *Selection Components*. Selection components filter the Viewsheet's underlying query or Data Block, and therefore affect *all* data view elements in the Viewsheet that use the same query or Data Block. In contrast, a condition that you place on a particular component affects only that component.

See Also
Filtering a Table and *Filtering a Chart* in *Interacting and Collaborating*, for an alternative method of filtering tables and charts.
Filtering, in *Advanced Data Mashup*, for information on setting Data Block conditions.
Selection Components, for dynamic filtering of data view components.

4.2.1 Creating a Filter Condition

To apply a condition to a data view or output component, follow these steps:

1. Right-click the element, and select 'Condition' from the context menu. This opens the 'Condition' dialog box.



2. Select the operand and type of comparison from the menus at the top of the dialog box, e.g., "`[Company][is not][one of]`"



3. From the 'Change Value Type' popup menu (triangle), select 'Value', 'Field', 'Variable', or 'Session Data'.

   a. If you select the 'Field' option, in the adjoining menu, choose the column for comparison. (The two columns will be compared row by row; rows that satisfy the condition will be preserved.)

   b. If you select the 'Value' option, either enter the comparison value into the provided text field, or select 'Browse Data' to choose the comparison value from a list.

   c. If you select the 'Variable' option, select one of the *Input Components* listed in the menu to provide the value at runtime. (See *Dynamic Properties* for further details.)

   d. If you select the 'Expression' option, press the 'js'/'sql' button to select JavaScript or SQL syntax, and then press the 'Edit' button. This opens the Script Editor where you can enter your expression. (See *Adding Property Script (Expressions)* in *Advanced Dashboard Scripting* for details on using expressions.)

   e. If you select the 'Session Data' option, choose one of 'User', 'Roles', or 'Groups'. These parameters return information about the user who is currently viewing the Viewsheet; respectively, the user name, the array of roles to which the user belongs, and the array of groups to which the user belongs.

Note that the adjacent menu automatically chooses the 'equal to' or 'one of' operator to match your selection.

4. When you have fully specified the condition, click 'Append'.

5. To create a compound condition, repeat Steps 2-4 above, using the 'and' and 'or' operators to compose the condition clauses. See *Modifying a Filter/Highlight Condition* for more details.

6. Click 'OK' to exit the 'Condition' dialog box.

The component will only display data that is compatible with the selections and satisfies the condition.

See Also

Filtering a Table and *Filtering a Chart* in *Interacting and Collaborating*, for an alternative method of filtering.

Filtering Operators, in *Advanced Data Mashup*, for explanation of the operators.

4.3 Hyperlinks

You can add hyperlinks to Data View component and Output components. (Data View components include Tables, Crosstabs, and Charts. Output components include Text, Image, Gauge, Thermometer, Sliding Scale, and Cylinder.)

Do not attach hyperlinks to a component that already has an onClick script.

Hyperlinks can target reports, Viewsheets, or web pages. When parameters are associated with a hyperlink, the parameter values are automatically passed to the targeted report, Viewsheet, or web page. Parameters can represent any of the attributes (columns) bound to the table or chart.

Note: You can also add hyperlinks to Viewsheets from within the Data Modeler. See *Auto-Drilldown*, in *Advanced Data Modeling*, for details on adding model-level and query-level hyperlinks.

The following example illustrates how to add hyperlinks to a table. Adding hyperlinks to other components is the same.

See Also

Adding Hyperlinks to a Table, to access the 'Hyperlink' dialog box for tables.

Adding Hyperlinks to a Chart, to access the 'Hyperlink' dialog box for charts.

Adding Hyperlinks to Output Components, to access the 'Hyperlink' dialog box for output components.

4.3.1 Creating a Hyperlink to a Report or Viewsheet

In this example, you will create a Viewsheet containing the 'Customer' table from 'Order Model', and add hyperlinks to the 'Company' attribute. The 'Company' attribute contains the names of companies, and the company names will be hyperlinked to the appropriate company-specific information in the 'OrderList' report (in the `classes/inetsoft/demos` directory).

Walkthrough

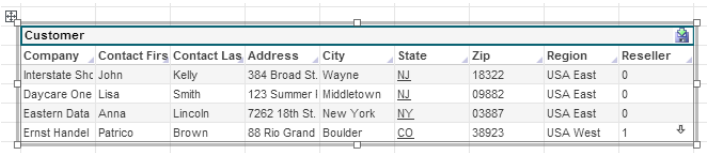
The 'OrderList' report is designed to accept five parameters, one of those being 'customer,' which represents the company name of the customer. You will configure the hyperlinks that you add to the 'Customer' > 'Company' attribute in the Viewsheet to pass the company name into the 'OrderList' report's 'customer' parameter.

1. Create a new Viewsheet based on the 'Order Model' data source.

2. From the **Component** panel, drag the entire 'Customer' entity into the Viewsheet grid. This creates a table with all the columns from the 'Customer' entity.

Double-click a column header to rename. Drag a column header to move the column.

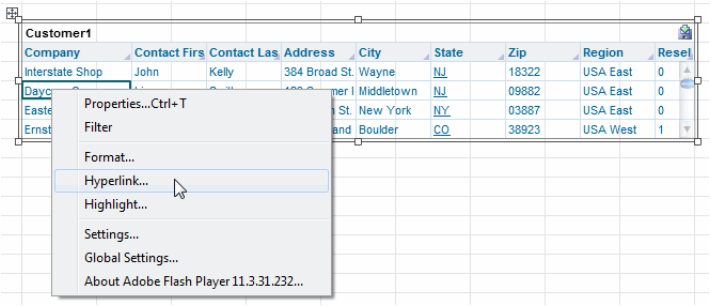
3. Rename and rearrange (if necessary) the columns as shown below:



4. Save the new Viewsheet as ‘HyperlinkExampleView’.

Add hyperlinks to the names in the ‘Company’ column. These hyperlinks will pass the company name as parameter to the ‘OrderList’ report. Follow the steps below:

5. Right-click any cell in the ‘Company’ column, and select the ‘Hyperlink’ option.



6. This opens the ‘Hyperlink’ dialog box for the ‘Company’ column.

7. In the ‘Hyperlink’ dialog box, click the ‘Asset Link’ radio button.

8. Select ‘OrderList’ from the ‘Link’ menu.

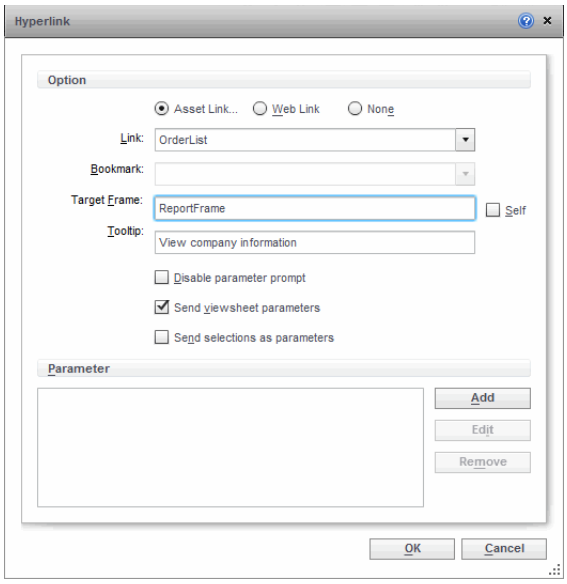
9. Optional: Enter a tooltip in the ‘Tool Tip’ field. (In the example we use “View company information” as the tooltip.)

   The tooltip is displayed when the end-user hovers the mouse over a hyperlink. If there are multiple hyperlinks for a given attribute (which may be due to auto-drills added at the model level or query level), the tooltip is displayed when the end-user hovers the mouse over the name of the hyperlink in the menu of hyperlink options.

10. Deselect the ‘Self’ option, and enter the name “ReportFrame” in the ‘Target Frame’ field.

```
Note: The 'Self' setting is not applied during preview in Visual
Composer, and is not available for hyperlinked Viewsheets. (All
hyperlinked Viewsheets open in a separate browser tab.)
```

This will cause the hyperlinked report to open in a window called “ReportFrame” (i.e., this is the browser DOM ‘name’ attribute). If you want the report to open in the original Viewsheet frame, replacing the Viewsheet, leave the ‘Self’ checkbox selected.

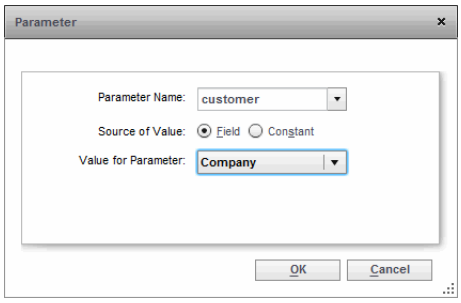


To delete an existing parameter, select the parameter and press ‘Remove’. To edit an existing parameter, press ‘Edit’.

11. Press the ‘Add’ button in the bottom panel. This opens the ‘Parameter’ dialog box.

12. Select the ‘customer’ parameter in the ‘Name’ menu. This is the name of the target parameter in the OrderList report.

13. Select the ‘Company’ field from the ‘Value’ menu. This is the name of the table column whose value should be passed into the report parameter when the user clicks the hyperlink.



14. Press ‘OK’.

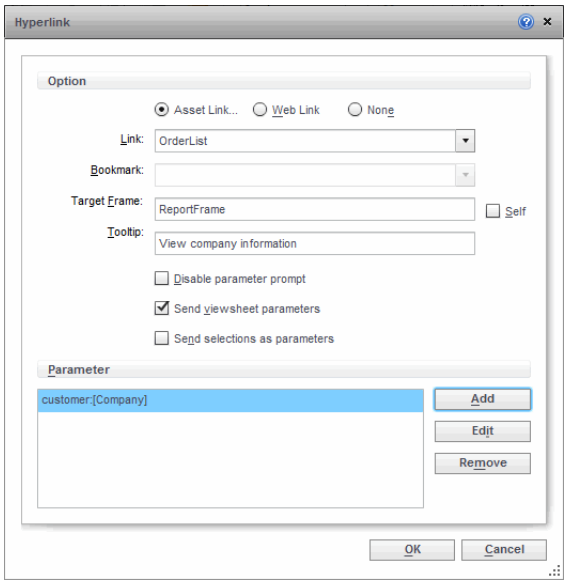
15. (Optional) Enable ‘Send viewsheet parameters’ to pass all existing viewsheet parameters to the drill-down report, Viewsheet, or URL. The viewsheet parameter names must exactly match those defined in the target report, Viewsheet, or URL. See *Defining and Editing*

Report Parameters in *Advanced Report Design*, and *parameter* in *Advanced Dashboard Scripting*, for more information.

16. (Optional) Select 'Disable parameter prompt' to suppress the parameter dialog box ordinarily produced by the drill-down report upon launch. (Prompting is automatically suppressed for parameters passed in the hyperlink itself.)

17. (Optional) Select 'Send selections as parameters' to send the values of any Viewsheet Selection components (Selection Lists, Range Sliders, etc.) to the target report. These values are passed into parameters in the target report that have the same name as the Selection component. For example, a Selection List named "EmployeeList" will pass its selected values into a parameter called 'EmployeeList' if one exists in the drill-down report.

18. Press 'OK' to complete the hyperlink configuration.

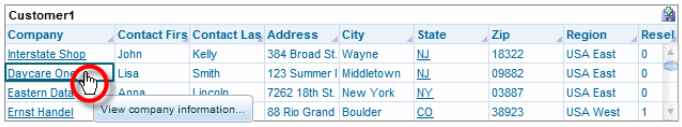


To display hyperlink underlining by default, set hyperlink.indicator=true in *sree.properties*.

19. Add any desired visual formatting to the hyperlinks (e.g., underlining).

20. Save the Viewsheet.

When the user views this Viewsheet in the User Portal, they will now see hyperlinks on all the names in the 'Company' column. (The 'State' column also shows hyperlinks because this attribute was pre-configured with an auto-drill.) Clicking one of the 'Company' hyperlinks launches the 'OrderList' report in a new browser window, and the 'OrderList' report now displays *only* the orders for the specific company clicked.



See Also

pviewsheet, in *Advanced Dashboard Scripting*, for information on how to access properties of a parent Viewsheet from within a hyperlinked report or Viewsheet.

parameter, in *Advanced Dashboard Scripting*, for information on reading and writing Viewsheet parameters.

Parameterization, in *Advanced Report Design*, for information on configuring report parameters.

4.3.2 Creating a Hyperlink to a Web Page

Walkthrough

You will continue the previous example (*Creating a Hyperlink to a Report or Viewsheet*) by changing the hyperlinks to target a *web page* rather than a report. The initial steps (up to and including Step 6) are the same as in the previous example.

1. In the ‘Hyperlink’ dialog box, click the ‘Web Link’ radio button.

You can also specify a table column to provide the URL links by selecting the desired column from the ‘Link’ menu.

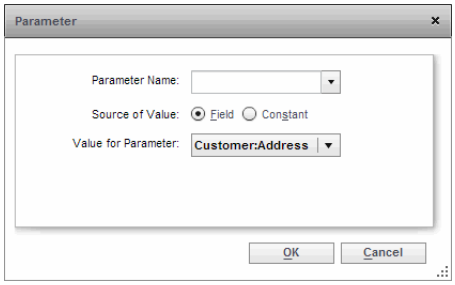
2. In the ‘Link’ field, enter the URL of the hyperlink destination with parameters omitted. For example, if the hyperlink is to pass a value as parameter to the Google search engine, the ‘Link’ field should specify “`http://www.google.com/search`”.

3. Optional: Enter a tooltip in the ‘Tool Tip’ field. (In the example we use “View company information” as the tooltip.)

```
Note: The 'Self' setting is not applied during preview in Visual
Composer.
```

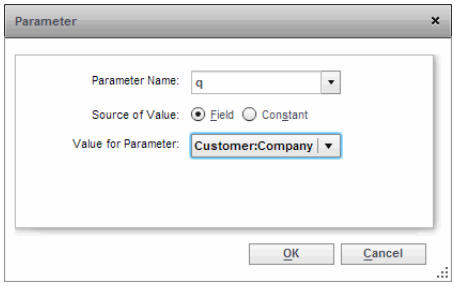
4. Optional: If you want the target web page to be displayed in a particular browser window, enter the name of the desired window in the ‘Target Frame’ field. (The specified name becomes the DOM ‘name’ attribute.) To open the target web page in the User Portal frame, select the ‘Self’ checkbox.

5. Press the ‘Add’ button. This opens the ‘Parameter’ dialog box.



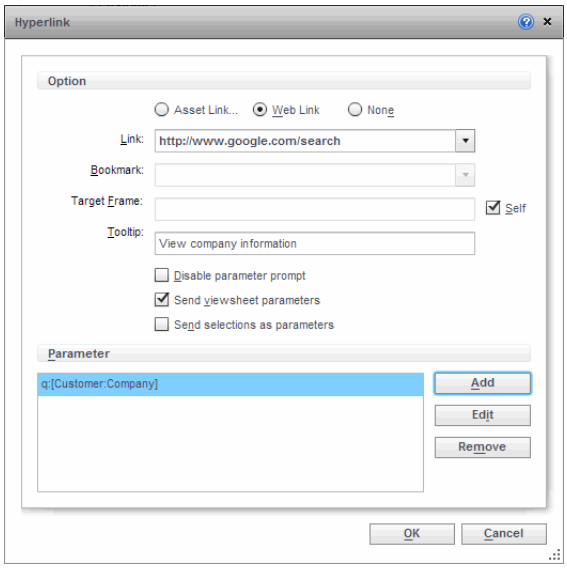
6. Enter ‘q’ in the ‘Parameter Name’ field. This name will be the parameter name used in the URL that is automatically constructed. (The parameter name “q” is the parameter name expected by the Google search engine.)

7. Select ‘Company’ from the ‘Value for Parameter’ menu.



8. Press ‘OK’ to close the ‘Parameter’ dialog box.

9. Press ‘OK’ to complete the hyperlink configuration. The final configuration of the ‘Hyperlink’ dialog box is shown below.



To display hyperlink underlining by default, set hyperlink.indicator=true in *sree.properties*.

10. Add any desired visual formatting to the hyperlinks (e.g., underlining).

11. Save the Viewsheet.

The hyperlinks constructed by this configuration will be of the form

```
http://www.google.com/search?q={CompanyName}
```

where `{CompanyName}` is the value of the 'Name' attribute clicked by the user.

4.3.3 Passing Multiple Parameters in a Hyperlink

The previous examples described how to pass a single parameter to a specified report or web page. It is also possible to pass multiple parameters in a hyperlink.

The procedure for creating multiple-parameter hyperlinks is the same as described in the previous section. If there are multiple attributes listed in the 'Value' panel of the 'Hyperlink' dialog box, these can all be selected and mapped to different parameter names.

If the hyperlinks are targeting a report, the multiple parameter names specified in the 'Hyperlink' window must be identical to the corresponding parameter names specified in the report. If the hyperlinks are targeting a web page, the multiple parameter names specified in the 'Hyperlink' window will be the parameter names used in the hyperlink URL, with parameter-value pairs separated by the ampersand (&) symbol, for example,

```
http://
remotehost?param1={value1}&param2={value2}&param3={value3}
```

where `param1`, `param2`, `param3` are the parameter names specified in the 'Hyperlink' window, and `{value1}`, `{value2}`, `{value3}` are the values of the attributes selected in the 'Value' panel.

4.4 Calculated Fields

You can perform calculations on existing data fields within a Viewsheet to create new "calculated" data fields. You can then use these calculated fields in the bindings of Charts, Crosstabs, and other elements.

This is especially useful for Viewsheets that are not based on Worksheet data sources (such as Viewsheets based on queries or data models) and do not have access to the Worksheet's computational tools. In such cases, the Viewsheet's calculated field feature provides an equivalent to the Worksheet's expression column feature. The Viewsheet's calculated field feature also aids in calculations that require access to *post-aggregate*

quantities, such as rates and ratios, which can be challenging to configure within a Worksheet.

Note: For Worksheet-based Viewsheets, you can make detail-level calculations within the Worksheet itself. (See *Creating an Expression/Formula Column* in *Advanced Data Mashup*.) To perform aggregate-level calculations, use the Viewsheet methods described below.

There are two different kinds of calculated field:

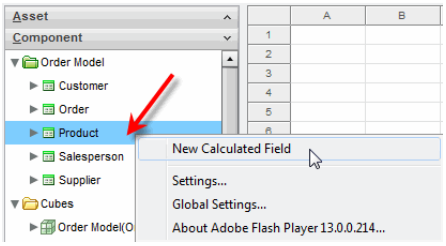
- **Calculated field based on detail data**: A calculated field based on detail data can be used as either a dimension or measure (see *Converting between Measure and Dimension*) in the same manner as any other data field. See *Creating a Calculated Field from Detail Data* for more information.
- **Calculated field based on aggregate data**: A calculated field based on aggregate data can be used only as a measure. This type of calculated field *cannot* be bound to *Selection Components* (Range Slider, etc.). See *Creating a Calculated Field from Aggregate Data* for more information.

The following sections explain how to create these two types of fields.

4.4.1 Creating a Calculated Field from Detail Data

To create a new calculated field from detail data, follow the steps below:

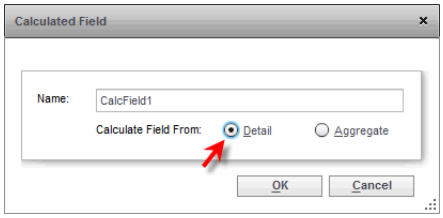
1. In the Viewsheet **Component** panel, right-click on the Data Block, data model, or query to which you want to add the calculated field. Select ‘New Calculated Field’ from the context menu.



   This opens the ‘Calculated Field’ dialog box.

Note: The selection of 'Detail' or 'Aggregate' is a one-time choice. You cannot convert between these two types at a later time.

2. In the ‘Name’ field, enter a name for the new calculated field.



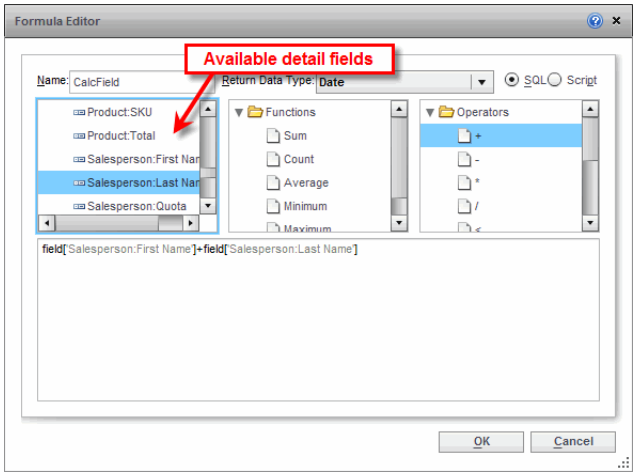
3. Select the ‘Detail’ option, and click ‘OK’. This opens the Formula Editor.

4. In the ‘Return Data Type’ menu, select the data type that the calculated field will return.

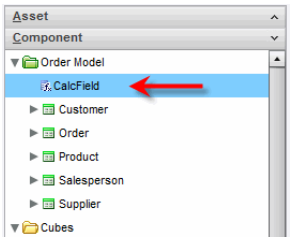
5. Enable the ‘SQL’ option to construct the formula using SQL syntax. Disable the ‘SQL’ option to construct the formula using JavaScript syntax.

Click on an available detail field in the panel to add it to the expression.

6. Enter the desired calculation into the formula field at the bottom of the dialog box. The available detail fields are listed in the top-left panel.



7. Click ‘OK’ to close the Formula Editor. This adds the new calculated field to the data source tree in the **Component** panel.



You can now bind this calculated field to a Viewsheet component (*Output Components*, *Data View Components*, etc.) in the same way that would bind any other data field.

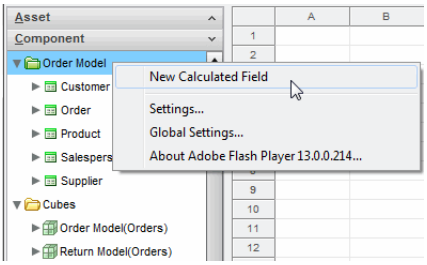
Example: Detail-Level Calculation

Consider a Viewsheet that is based on the sample 'Order Model' data model. In this example, you will analyze how the average difference between the 'Number in Stock' and the 'Reorder Level' varies according to product category. Note that this difference ('Number in Stock' – 'Reorder Level') is not available as a predefined attribute within the data model. However, you can create this quantity in the Viewsheet as a calculated field.

Follow the steps below:

1. Create a new Viewsheet based on the 'Order Model' data model. (See *Creating a New Viewsheet* for more information about creating a Viewsheet.)

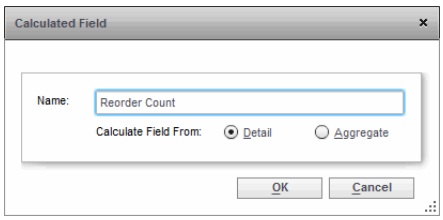
2. Right-click the 'Order Model' node in the **Component** panel, and select 'New Calculated Field' from the context menu.



   This opens the 'Calculated Field' dialog box.

3. In the 'Name' field of the 'Calculated Field' dialog box, enter the text "Reorder Count".

4. Select the 'Detail' option and click 'OK'. This opens the Formula Editor.



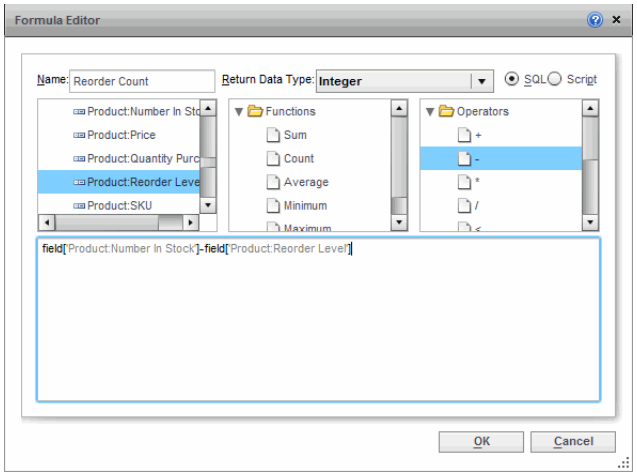
5. In the Formula Editor, follow the steps below:

a. In the top-left panel, click the ‘Product:Number In Stock’ field to enter the field into the expression.

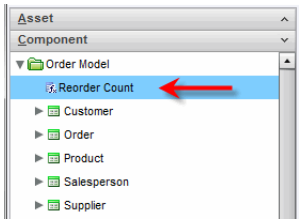
b. Type a minus (“–”) symbol, or select this symbol from the top-right ‘Operators’ panel.

c. In the top-left panel, click the ‘Product:Reorder Level’ field to enter the field into the expression.

d. In the ‘Return Data Type’ menu, select ‘Integer’.



e. Click ‘OK’ to close the Editor. This adds the new ‘Reorder Count’ calculated field to the ‘Order Model’.



6. From the **Component** panel, drag a Chart component into the Viewsheet.

7. Click the ‘Edit’ button on the new Chart to open the Chart Editor.

8. In the **Data Source** panel, expand the ‘Dimensions’ folder under the ‘Product’ entity.

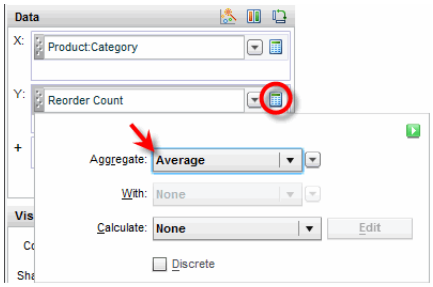
9. Drag the ‘Product:Category’ field from the **Data Source** panel to the ‘X’ region in the **Data** panel.

```
The 'Reorder Count' calculated field appears by default under the
'Measures' folder because it has a numerical 'Return Data Type'.
```

10. In the **Data Source** panel, expand the ‘Measures’ folder.

11. Drag the ‘Reorder Count’ field from the **Data Source** panel to the ‘Y’ region in the **Data** panel.

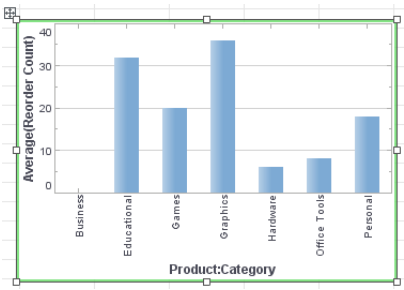
12. Click the ‘Edit Measure’ button next to the ‘Y’ region in the **Data** panel, and select ‘Average’ from the ‘Aggregate’ menu.



13. Click the green ‘Apply’ button.

    The Chart Y-axis now displays the average of the calculated difference between ‘Number in Stock’ and ‘Reorder Level’ on a category-by-category basis.

14. Resize the Chart so that the Y-axis title is fully displayed.

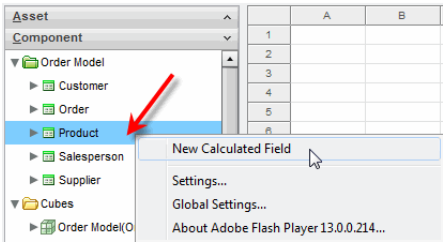


¢

4.4.2 Creating a Calculated Field from Aggregate Data

To create a new calculated field from aggregate data, follow the steps below:

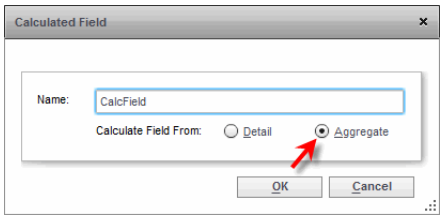
1. In the Viewsheet **Component** panel, right-click on the Data Block, data model, or query to which you want to add a calculated field. Select ‘New Calculated Field’ from the context menu.



This opens the ‘Calculated Field’ dialog box.

Note: The selection of 'Detail' or 'Aggregate' is a one-time choice. You cannot convert between these two types at a later time.

2. In the ‘Name’ field, enter a name for the new calculation field.

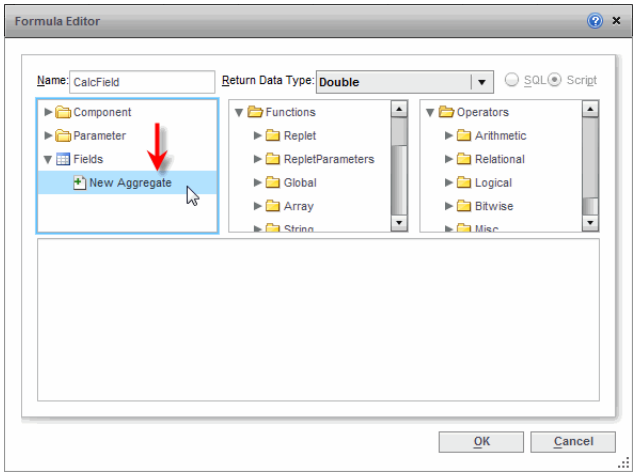


3. Select the ‘Aggregate’ option and click ‘OK’. This opens the Formula Editor.

The ‘SQL’ option is disabled because aggregate-based calculations require JavaScript syntax.

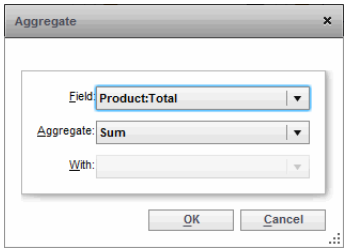
4. In the ‘Return Data Type’ menu, select the data type that the calculated field will return.

5. In the top-left panel, click the ‘New Aggregate’ button.



This opens the ‘Aggregate’ dialog box.

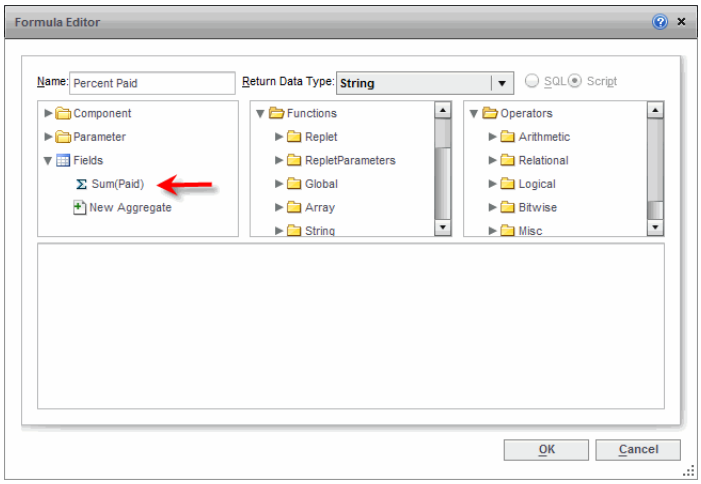
6. In the ‘Field’ menu of the ‘Aggregate’ dialog box, select the field that you wish to aggregate for purposes of the calculation.



7. In the ‘Aggregate’ menu, select the desired aggregation method. (See *Aggregation Options* for information about the available methods.)

8. For a bivariate aggregation method, select the second field from the ‘With’ menu.

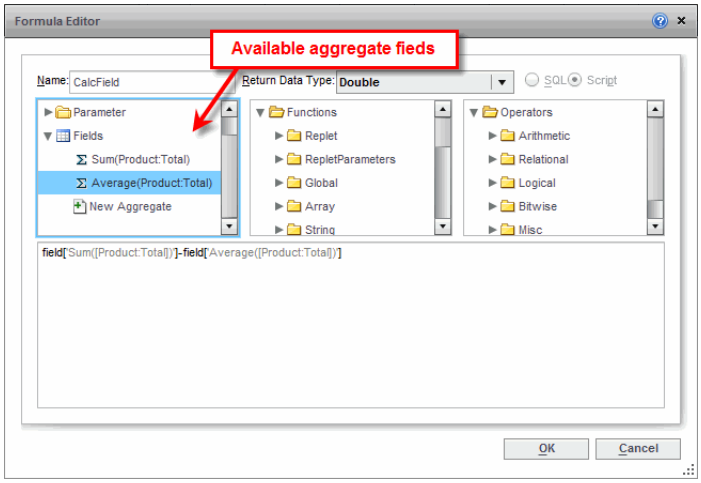
9. Click ‘OK’ to close the ‘Aggregate’ dialog box. The Formula Editor now displays the newly defined aggregate quantity in the top-left panel.



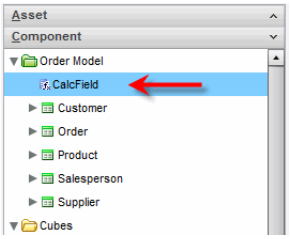
10. (Optional) Click the ‘New Aggregate’ button again, and repeat the above steps to add as many additional aggregates as desired.

Click an aggregate field in the top-left panel to add it to the expression.

11. Once you have defined the required aggregates in the top-left panel, enter the desired calculation (using the aggregate fields) in the formula field at the bottom of the dialog box.



12. Click ‘OK’ to close the Formula Editor. This adds the new calculated field to the data source tree in the **Component** panel.



`You cannot bind an aggregate-based calculated field to` *`Selection Components`*`.`

You can bind this calculated field to *Output Components* or *Data View Components* in the same way that would bind any other data field.

Example: Aggregate-Level Calculation

Consider a Viewsheet that is based on the sample ‘Sales Explore’ Worksheet. In this example, you will analyze how the *percentage of paid orders* varies according to different dimensions (company, state, date, etc.).

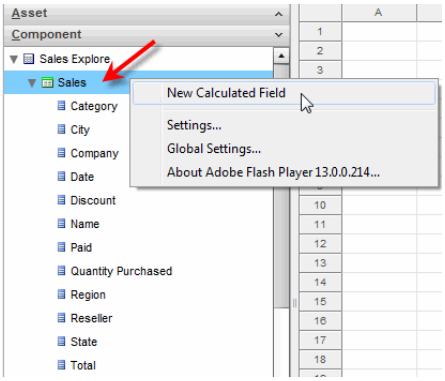
Observe that “percentage of paid orders” for a given group is an *aggregate* quantity. It represents the ratio of paid orders to total orders for the group, and therefore requires a count (aggregate) of both the paid orders and total orders for every individual group. These counts are not static, but will vary dynamically based on the requested dimension (state, company, date, etc.).

In many cases it is difficult to configure an underlying Worksheet to produce these aggregate results in a manner conducive to dynamic Viewsheet analysis. Though a Worksheet provides all of the required aggregation features, multi-dimensional aggregations within a Worksheet often require the creation of multiple Data Blocks, and this ultimately limits the interactivity of the Viewsheet. Therefore, it is best to implement aggregate-based calculations at the Viewsheet level.

Follow the steps below to create a Viewsheet that implements aggregate calculations to express the percentage of paid orders for various groupings.

1. Create a new Viewsheet based on the 'Sales' > 'Sales Explore' Data Worksheet. (See *Creating a New Viewsheet* for information about creating a Viewsheet.)

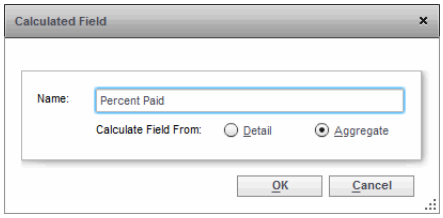
2. In the Viewsheet **Component** panel, right-click the 'Sales' data block and select 'New Calculated Field' from the context menu.



   This opens the 'Calculated Field' dialog box.

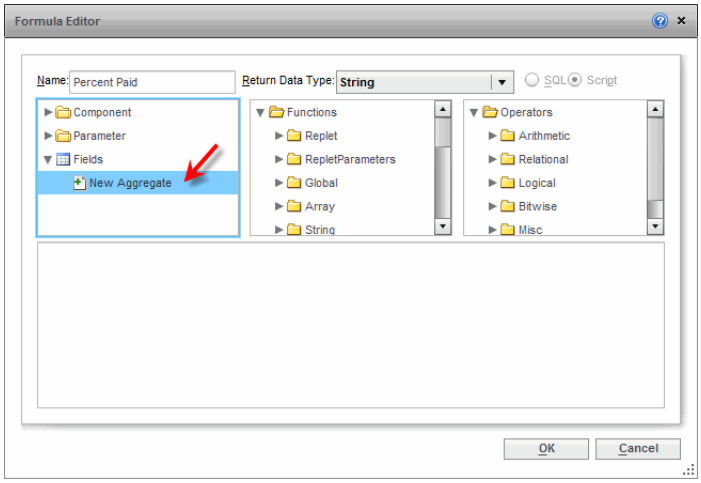
3. In the 'Name' field of the 'Calculated Field' dialog box, enter the text "Percent Paid".

4. Select the 'Aggregate' option and click 'OK'. This opens the Formula Editor.



5. In the Formula Editor, follow the steps below:

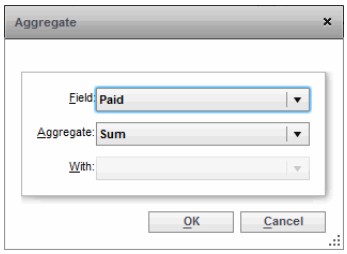
   a. In the top-left panel, expand the 'Fields' node, and click 'New Aggregate'.



This opens the 'Aggregate' dialog box.

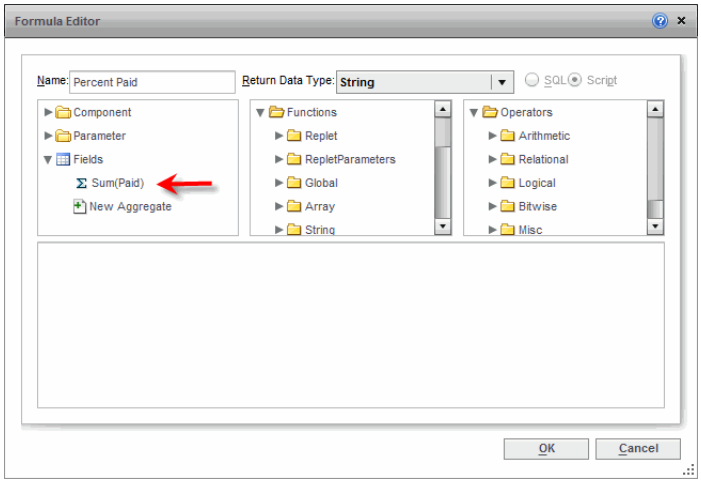
The 'Paid' field contains a "1" (order paid) or a "0" (order not paid). This sum therefore provides the absolute number of paid orders.

b. In the 'Field' menu of the 'Aggregate' dialog box, select the 'Paid' field. This is the field whose values you will sum to obtain the number of paid orders.



c. In the 'Aggregate' menu, select 'Sum'.

d. Click 'OK' to close the 'Aggregate' dialog box. The Formula Editor displays the newly defined "Sum(Paid)" aggregate field under the 'Fields' node in the top-left panel.

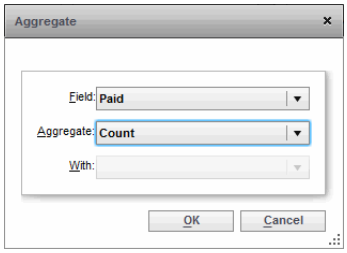


e. Click ‘New Aggregate’ again.

f. In the ‘Field’ menu of the ‘Aggregate’ dialog box, again select the ‘Paid’ field.

The ‘Count’ provides the total number of paid and unpaid orders.

g. In the ‘Aggregate’ menu, select ‘Count’ and click ‘OK’.



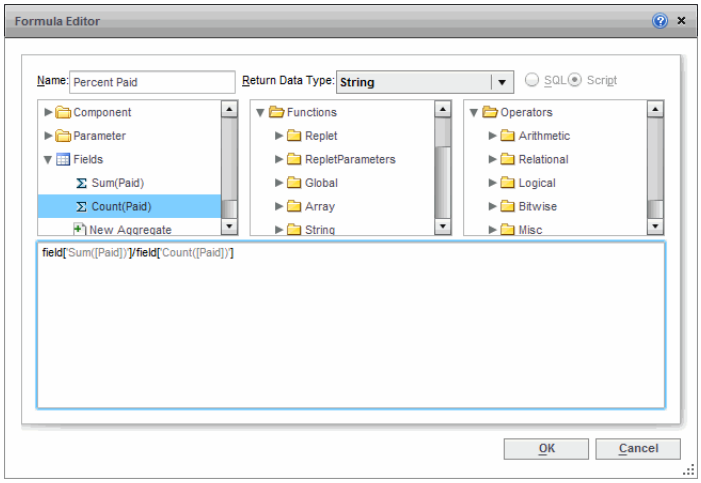
The Formula Editor displays the newly defined ‘Count(Paid)’ aggregate field in the top-left panel together with the existing ‘Sum(Paid)’ field.

h. In the top-left panel, click on the ‘Sum(Paid)’ field to enter the field into the expression.

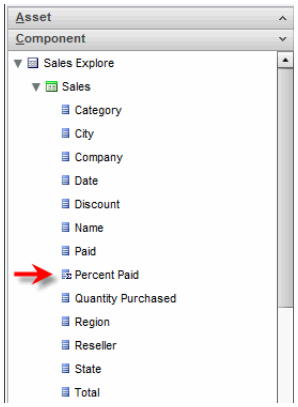
i. Type a division (“/”) symbol, or select this symbol from the top-right ‘Operators’ panel (under the ‘Arithmetic’ group).

j. In the top-left panel, click on the ‘Count(Paid)’ field to enter the field into the expression.

k. In the ‘Return Data Type’ menu, select ‘Float’.



l. Click ‘OK’ to close the Editor. This adds the new ‘Percent Paid’ calculated field to the ‘Sales’ data block.



6. From the **Component** panel, drag a Chart component into the Viewsheet.

7. Click the ‘Edit’ button on the new Chart to open the Chart Editor.

8. In the **Data Source** panel, expand the ‘Dimensions’ folder under the ‘Sales’ data block.

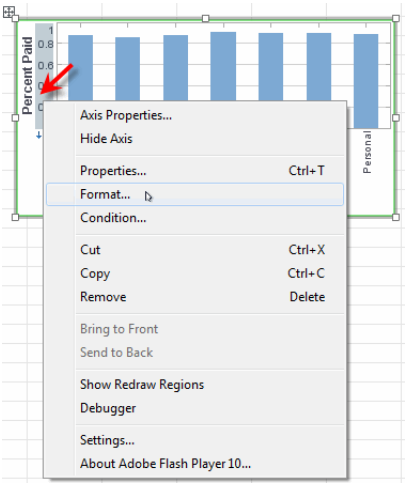
9. Drag the ‘Category’ field from the **Data Source** panel to the ‘X’ region in the **Data** panel.

```
Note: An aggregate-based calculated field is always considered a
measure.
```

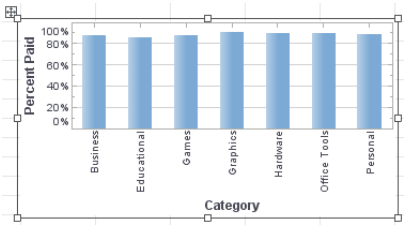
10. In the **Data Source** panel, expand the ‘Measures’ folder.

11. Drag the ‘Percent Paid’ field from the **Data Source** panel to the ‘Y’ region in the **Data** panel. The Chart Y-axis now displays the fraction of orders paid on a category-by-category basis.

12. To change the Y-axis labels to a “percent” representation, right-click the axis labels, and select ‘Format’ from the context menu. This opens the ‘Format’ dialog box.



13. In the ‘Format’ dialog box, click the **Format** tab, and select the ‘Percent’ option. Click ‘OK’ to close the dialog box. This displays the labels in percent format.



You can use the Chart Editor to bind other dimensions to the X-axis. For example, drag ‘Name’ from the **Data Source** panel to the ‘X’ region of the **Data** panel to analyze the percent paid according to product name. Drag both the ‘State’ and ‘City’ fields to the ‘X’ region to analyze percent paid on a geographical basis.

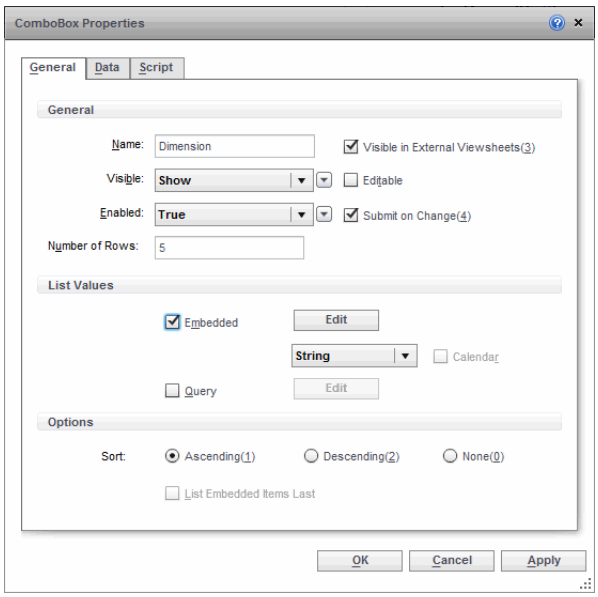
If you provide the end user with access to the Chart Editor (see *Enabling End-User Chart and Crosstab Editing*), they can use exactly the same method to analyze the data. However to make the analysis even easier for the user, you can add a menu that allows the user to directly select the desired dimension for the X-axis. To do this, continue with the steps below:

14. Close the Chart Editor. (Click the red ‘X’ in the top-right corner.)

15. From the Viewsheet **Component** panel, drag a ComboBox component into the Viewsheet.

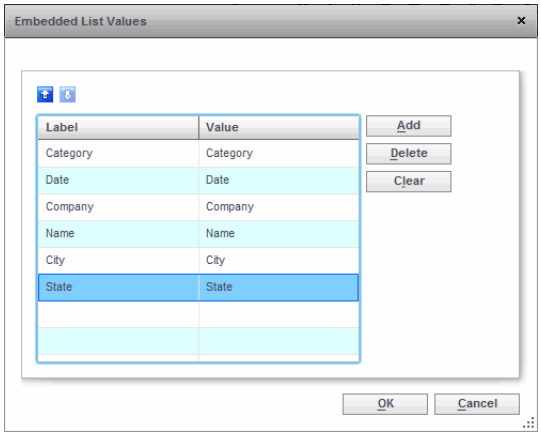
16. Right-click on the new ComboBox, and select ‘Properties’ from the context menu. This opens the ‘ComboBox Properties’ dialog box.

17. In the ‘Name’ field, enter ‘Dimension’.



18. In the ‘List Values’ region, select the ‘Embedded’ option, and click the ‘Edit’ button. This opens the ‘Embedded List Values’ dialog box.

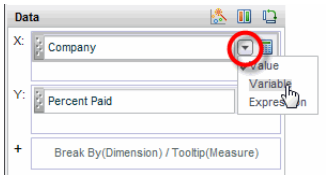
19. Enter the following labels/values: ‘Category’, ‘Date’, ‘Company’, ‘Name’, ‘City’, and ‘State’.



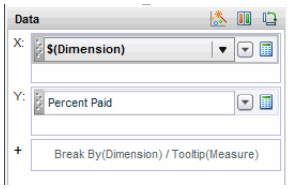
20. Click ‘OK’ to close the ‘Embedded List Values’ dialog box. Click ‘OK’ to close the ‘ComboBox Properties’ dialog box.

21. Click the ‘Edit’ button on the Chart to reopen the Chart Editor.

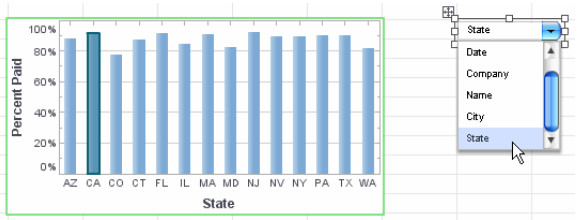
22. In the **Data** panel, click the arrow button next to the ‘X’ region, and select ‘Variable’ from the menu. This enables the variable menu in the ‘X’ region.



23. From the variable menu, select the ‘$(Dimension)’ option. This will use the ComboBox to set the dimension displayed on the X-axis.



Explore the data by making different selections in the ComboBox. Investigate how percent paid varies according to dimensions such as state, city, category, date, etc. When you analyze the data along the ‘Date’ dimension, try drilling down to explore finer levels of resolution (e.g., month). See *Drilling Down into a Chart* for more details.



¢

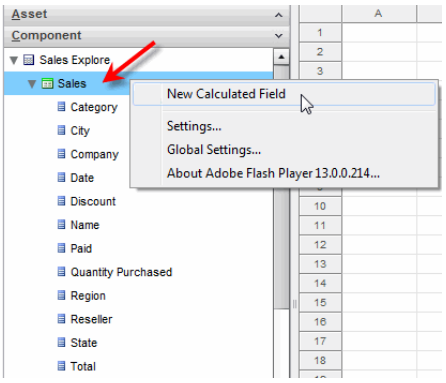
Example: Calculation with String Return Value

You can create an aggregate calculation to return a string-valued measure. You can then bind this measure to a ‘Visual’ region (color, shape, size) to produce a categorical visual grouping.

Follow the steps below to create a Viewsheet that uses a string-valued measure to categorize groups as having a low average discount rate (less than 1.5%) or high discount rate (greater than 1.5%).

1. Create a new Viewsheet based on the ‘Sales’ > ‘Sales Explore’ Data Worksheet. (See *Creating a New Viewsheet* for information.)

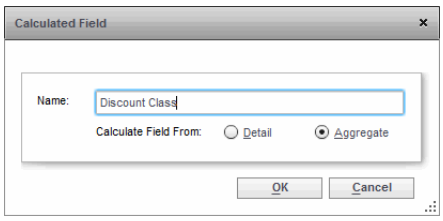
2. In the Viewsheet **Component** panel, right-click the ‘Sales’ data block and select ‘New Calculated Field’ from the context menu.



This opens the ‘Calculated Field’ dialog box.

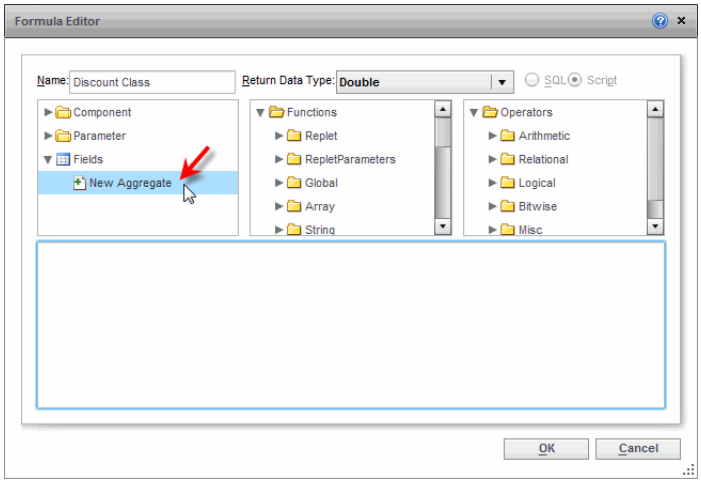
3. In the ‘Name’ field of the ‘Calculated Field’ dialog box, enter the text “Discount Class”.

4. Select the ‘Aggregate’ option and click ‘OK’. This opens the Formula Editor.



5. In the Formula Editor, follow the steps below:

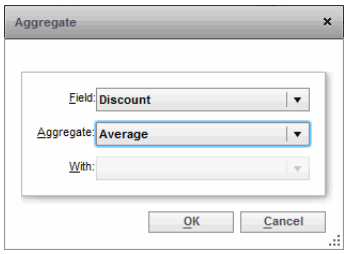
   a. In the top-left panel, expand the ‘Fields’ folder and click the ‘New Aggregate’ button.



This opens the ‘Aggregate’ dialog box.

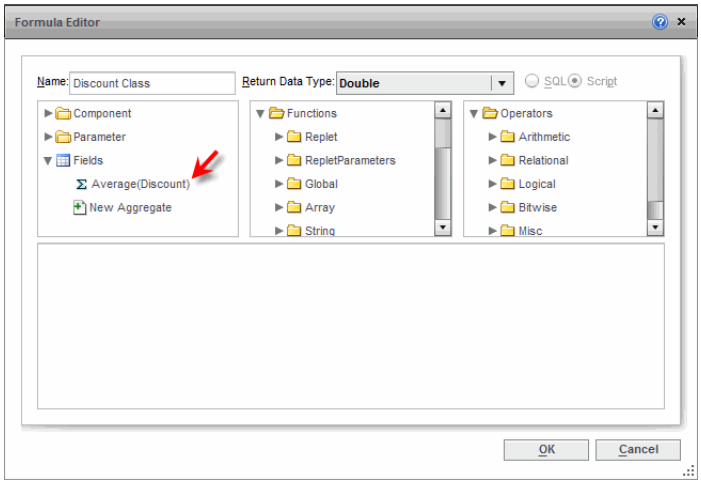
The ‘Paid’ field contains a “1” (order paid) or a “0” (order not paid). This sum therefore provides the absolute number of paid orders.

b. In the ‘Field’ menu of the ‘Aggregate’ dialog box, select the ‘Discount’ field. This is the field whose values you will average to obtain the average discount.



c. In the ‘Aggregate’ menu, select ‘Average’.

d. Click ‘OK’ to close the ‘Aggregate’ dialog box. The Formula Editor displays the newly defined “Average(Discount)” aggregate field in the top-left panel.

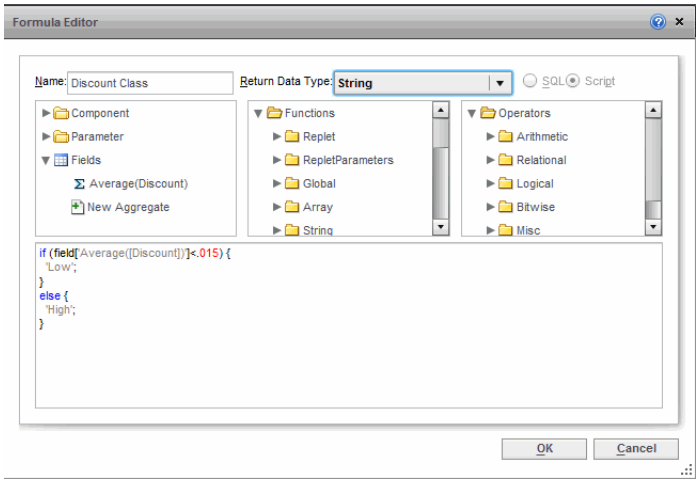


Hint: Click the 'Average(Discount)' field in the top-left panel to enter the field into the expression.

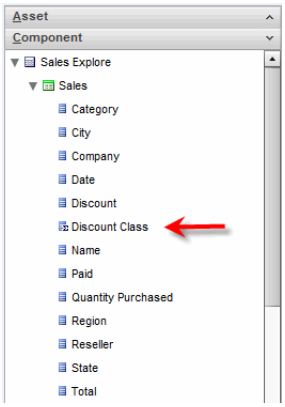
e. Enter the following script in the Script Editor:

```
if (field['Average([Discount])']<.015) {
  'Low';
}
else {
  'High';
}
```

f. In the 'Return Data Type' menu, select 'String'.



g. Click 'OK' to close the Editor. This adds the new 'Discount Class' calculated field to the 'Sales' data block.



6. From the **Component** panel, drag a Chart component into the Viewsheet.

7. Click the 'Edit' button on the new Chart to open the Chart Editor.

8. In the **Data Source** panel, expand the 'Dimensions' folder under the 'Sales' data block.

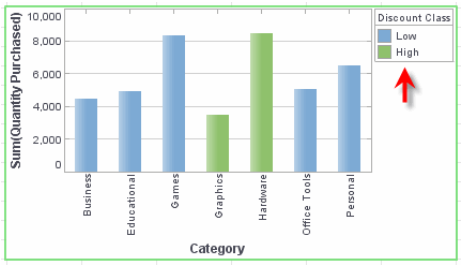
9. Drag the 'Category' field from the **Data Source** panel to the 'X' region in the **Data** panel.

```
Note: An aggregate-based calculated field is always considered a
measure.
```

10. In the **Data Source** panel, expand the 'Measures' folder.

11. Drag the 'Quantity Purchased' field from the **Data Source** panel to the 'Y' region in the **Data** panel.

12. Drag the 'Discount Class' field from the **Data Source** panel to the 'Color' region in the **Visual** panel. This discriminates groups based on the string values returned by the 'Discount Class' calculated field.

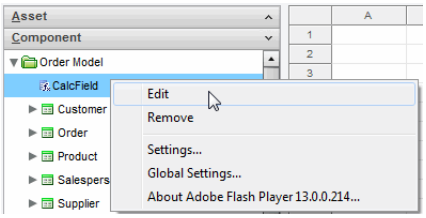


¢

4.4.3 Editing a Calculated Field

To edit an existing calculated field, follow the steps below:

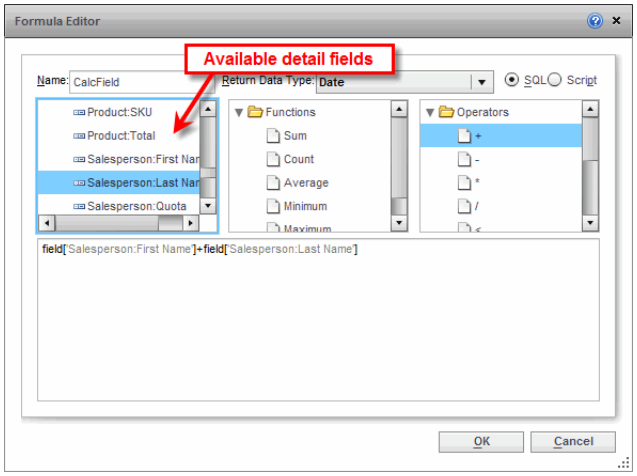
1. In the Viewsheet **Component** panel, right-click the calculated field in the data source tree, and select ‘Edit’ from the context menu.



You cannot convert a detail-based calculated field into a aggregate-based calculated field, or vice versa.

This opens the Formula Editor to display the existing formula.

2. Edit the formula as desired, and click ‘OK’.



To remove an existing calculated field, right-click on the calculated field in the data source tree in the **Component** panel, and select ‘Remove’ from the context menu.

5 Selection Components

Selection components provide both input and output capability, and are therefore crucial for analysis.

Selection controls are optimized for performance. They do not modify the original dataset, and do not cause re-execution of queries.

- Selection components control and filter data for data view and output components. By adding multiple selection elements, you exponentially increase possible views.
- Selections interact with one another to provide you with instant feedback on which choices are consistent and which are incompatible.

This dual nature of selections makes them an indispensable analytic component for most Viewsheets.

See Also
Filtering a Table and *Filtering a Chart* in *Interacting and Collaborating*, for an approach to filtering data without separate Selection components.
Filter Conditions, to set fixed conditions on data view and output components.
Output Components, for information about graphical data representation.
Input Components, for information on manipulating variables.
Data View Components, for information about tables and graphs.

5.1 Selection List

A Selection List displays every unique value from a data field. By selecting one or more of these values, the end-user can filter the data displayed by the Viewsheet, both within output elements and other selection elements.

See Also
Interacting and Collaborating, for information on using dashboards.
Using Selection Lists and Trees, for details on Selection element symbols.
Selection Tree, for information on providing nested selection options.

5.1.1 Creating a Selection List

There are four ways to create a Selection List, discussed in the following sections.

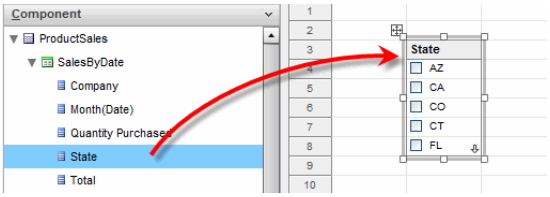
Creating Selection List from Data Block

To create a Selection List, follow these steps:

1. Expand a Data Block (table) or cube dimension in the Component tree.

Note: When binding hierarchical cube dimensions, Selection Lists will include all hierarchical components. To avoid this, ask an administrator to set the property *olap.table.originalContent=false* in *sree.properties*.

2. Select *one column* from the Data Block, or *one level* from the cube dimension, and drag it onto the Viewsheet grid.



This creates a Selection List based on the data in the selected column or dimension.

Creating Selection List from Selection List Component

To create a Selection List follow these steps:

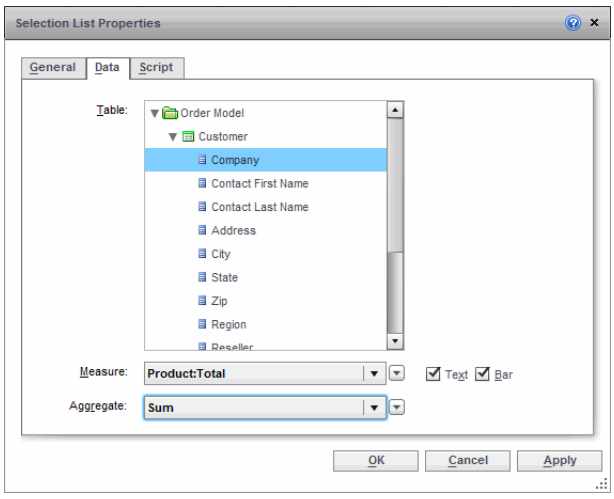
1. Drag a Selection List element from the Component tree onto the Viewsheet grid. This creates an empty Selection List element.

You can also drag and drop a single column directly onto the empty Selection List element.

2. Right-click on the Selection List, and select ‘Properties’ from the context menu. This opens the ‘Selection List Properties’ dialog box.

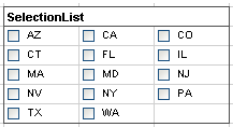
3. Select the **Data** tab.

4. In the ‘Table’ list, expand a Data Block or cube, and select the column or dimension that should be used to populate the Selection List.



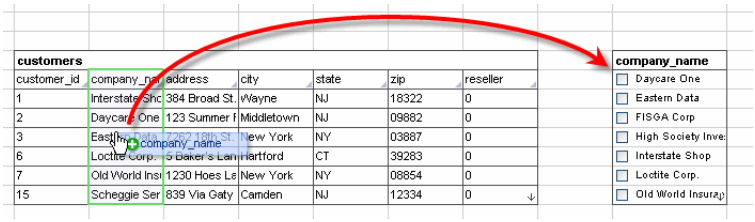
5. Click ‘OK’.

This creates a Selection List based on the data in the selected column or dimension. Each unique item in the column or dimension is represented as a distinct selection box in the Selection List.



Creating Selection List from a Table Column

To create a Selection List from a table column, click anywhere in the table column, and drag the column to an empty region of the grid.



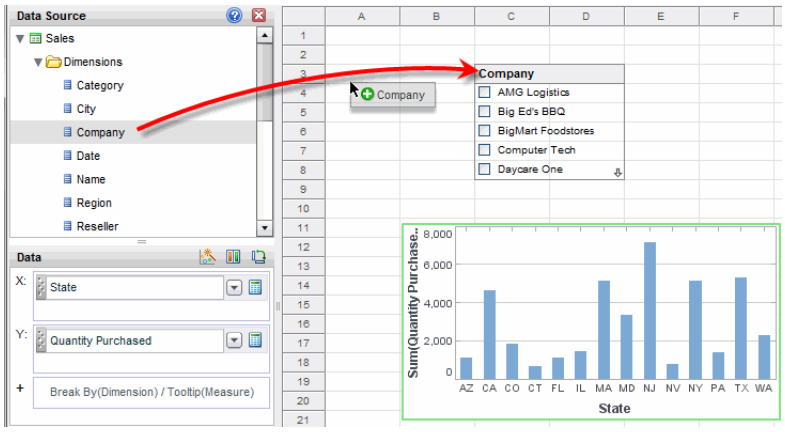
This creates a new Selection List based on the values in the column.

Creating Selection List from Chart Data Source

To create a Selection List from a chart Data Source tree, follow these steps:

1. Expand a ‘Dimension’ node in the Data Source tree. (This can belong to either a Data Block or cube.)

2. Select *one column* from the ‘Dimension’ node, and drag it onto the Viewsheet grid.



This creates a Selection List based on the data in the selected field.

See Also

Creating a Selection Tree from Chart Data Source, for information on the Selection Tree.

Creating a Chart, for information on using the Chart Editor.

5.1.2 Selection List Properties

The Selection List component's 'Properties' dialog box provides **General** and **Data** tabs. The next sections discuss the component-specific properties available under these tabs.

See Also

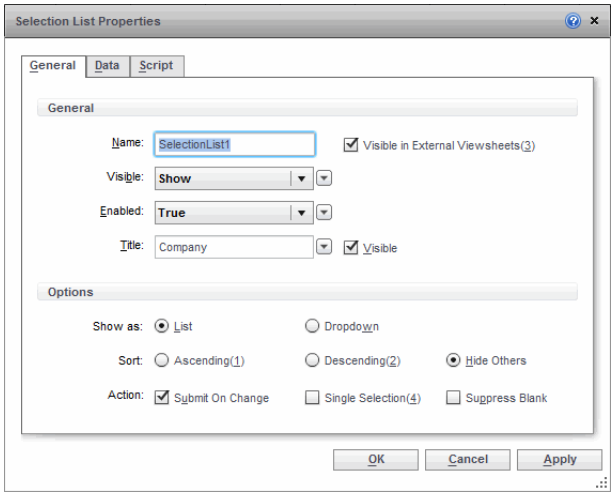
Properties Dialog Box, for information about common component properties.

Format Dialog Box, for information on the 'Format' dialog box.

Selection List Properties: General Tab

The **General** tab in the 'Selection List Properties' dialog box provides the following Selection List-specific properties.

Show as	The appearance of the Selection List: **List**: A scrollable list **Dropdown**: An expandible dropdown box. The 'List Height' sub-option sets the number of rows to which the box should expand.
Sort	The sort order for the values in the Selection List: **Ascending**: Display all values in alphabetical or increasing order, including incompatible selections. **Descending**: Display all values in reverse-alphabetical or decreasing order, including incompatible selections. **Hide Others**: Display values in alphabetical or increasing order, but hide incompatible selections under the label 'Others'.
Action	**Submit on Change**: Immediately update data in the Viewsheet when any selection is made. If 'Submit on Change' is disabled, the Selection List only updates data in the Viewsheet when the user clicks 'Apply'. **Single Selection**: Allows only one item to be selected at a given time. (For a Selection Tree, this allows only one item on the lowest level to be selected.) **Suppress Empty Value**: Removes empty values (caused by nulls in the data set) from the Selection List.



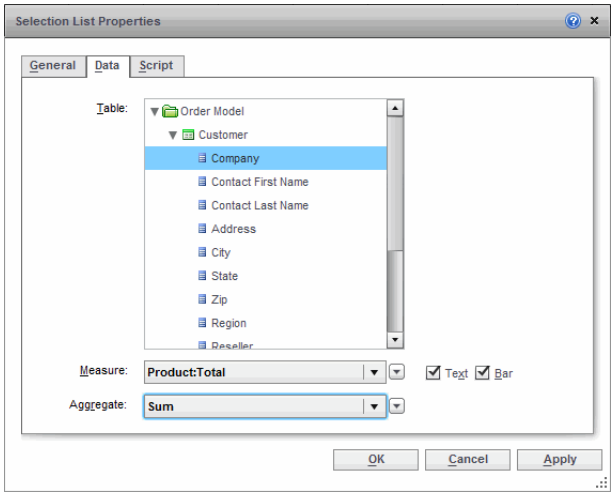
See Also

General Tab, for information about common properties in the **General** tab.

Selection List Properties: Data Tab

The **Data** tab in the 'Selection List Properties' dialog box provides the following Selection List-specific properties.

Table	The column or dimension that should be used to populate the Selection List
Measure	The field to aggregate and display next to the Selection List values.
Aggregate	The method of aggregation to be used for the 'Measure' field.
Text	Specifies that the aggregated measure is displayed as plain text.
Bar	Specifies that the aggregate measure is displayed graphically as a proportional bar.



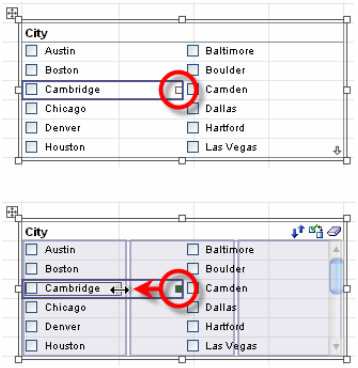
5.1.3 Changing the Column Layout in a Selection List

Columns within a Selection List always have equal width. This width is determined by the overall width of the Selection List component (see *Resizing a Component*) and by the number of columns specified.

To specify the number of columns in the Selection List component, follow the steps below:

1. Click a cell in any column to select the cell. (The right side of the selected cell displays a handle).

2. Drag the cell handle to the left or right to increase or decrease the number of columns.



See Also

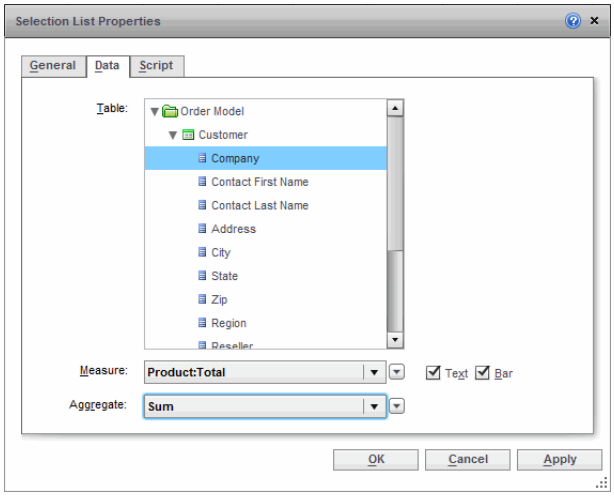
Resizing the Viewsheet Grid, for information on changing the global grid spacing.

5.1.4 Displaying Aggregate Data in Selection List

To display summarized data next to the dimension values in a Selection List or Selection Tree, follow the steps below:

1. Create the Selection List (or Selection Tree) as desired.

2. Right-click the Selection List, and select ‘Properties’ from the context menu. This opens the ‘Selection List Properties’ dialog box.



3. From the ‘Measure’ menu, select the field that you wish to summarize.

4. From the ‘Aggregate’ menu, select the method of summarization. (See *Crosstab Aggregation Methods* for more information about the methods.)

5. (Optional) Enable the ‘Text’ option to display the aggregate values as plain text next to the corresponding dimension values.

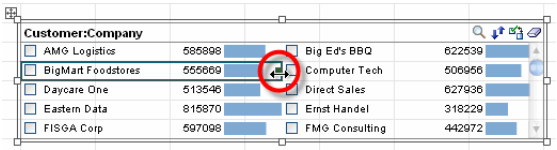
6. (Optional) Enable the ‘Bar’ option to graphically display the aggregate values as proportional bars next to the corresponding dimension values.

7. Click ‘OK’ to close the dialog box. The summarized values are now displayed next to the Selection List dimension values.

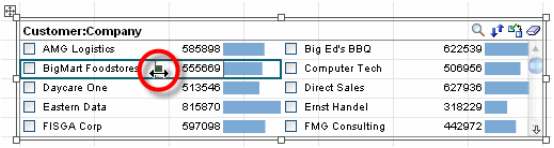
You can also adjust the Viewsheet grid width to facilitate display. See *Resizing the Viewsheet Grid*.

8. (Optional) Improve the appearance of the Selection List by making the following adjustments to the layout:

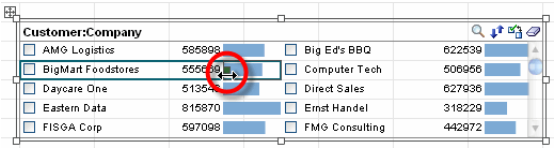
   a. Adjust the column layout to provide the needed space by dragging the handle at the right side of column. See *Changing the Column Layout in a Selection List*.



   b. Adjust the width of the aggregate text region by dragging the handle at the left side of the text.



c. Adjust the width of the aggregate bar region by dragging the handle at the left side of the bar.



You can also use the format controls on the Viewsheet toolbar to adjust the text and bar formatting.

9. (Optional) To format the aggregate text, right-click on the aggregate text and select 'Format' from the context menu. This opens the 'Format' dialog box. (See *Format Dialog Box* for more information about the available options.)

10. (Optional) To format the aggregate bar, right-click on the aggregate bar and select 'Format' from the context menu. This opens the 'Format' dialog box. (See *Format Dialog Box* for more information about the available options.)

5.2 Selection Tree

A Selection Tree is a hierarchical Selection List, and displays selection choices in tree form. This allows the user to select values at multiple levels in the hierarchy. The selections that the user makes in a Selection Tree filter data displayed by *Output Components*, *Data View Components*, and other *Selection Components*. The Selection Tree is especially useful for data that is intrinsically hierarchical, such as cities and states, but it can also be used for independent data.

You can define a Selection Tree by specifying an arbitrary hierarchy among individual columns or by referencing predefined parent-child relationships (recursive hierarchy). See the following sections for details on these two approaches:

- *Creating a Selection Tree from Individual Columns*
- *Creating a Selection Tree from Recursive Hierarchy*

5.2.1 Creating a Selection Tree from Individual Columns

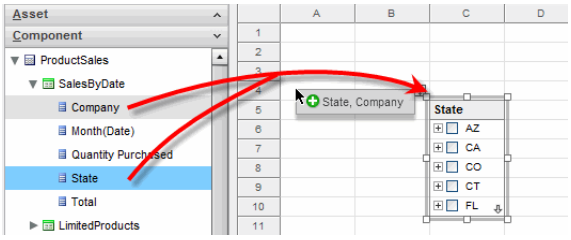
There are several ways to create a Selection Tree from individual columns. In most cases these columns should possess an implied inclusion relationship (e.g., cities and states) but they can also be independent.

The following sections discuss various methods for creating a Selection Tree from individual columns.

Creating a Selection Tree from Data Block Columns

To create a Selection Tree from Data Block columns, follow these steps:

1. Expand a Data Block (table) in the Component tree.
2. Ctrl-click or Shift-click to select *multiple columns* in the Data Block.
3. Drag the selected columns onto the Viewsheet grid.



This creates a Selection Tree based on the data in the selected columns, with the nesting order determined by the order in which you selected the fields in the Component tree.

See Also

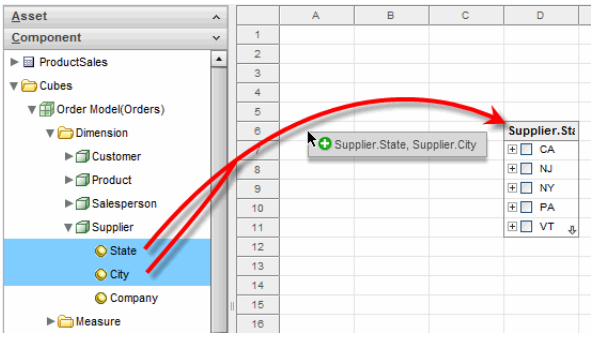
Adding Hierarchy Levels to a Selection Tree, to add levels to an existing Tree.
Selection Trees, for a walkthrough on using Selection Trees.
Using Selection Lists and Trees, for details on Selection element symbols.
Selection List, for information on creating non-hierarchical selection options.

Creating a Selection Tree from Cube Dimensions

To create a Selection Tree from cube dimension levels, follow these steps:

1. Expand a cube dimension in the Component tree.
2. Ctrl-click or Shift-click to select *multiple levels* in the cube dimension.
3. Drag the selected levels onto the Viewsheet grid.

To include *all* levels of a dimension, simply drag the dimension node onto the Viewsheet.



This creates a Selection Tree based on the data in the selected levels, with the nesting order determined by the order in which you selected the levels in the Component tree.

See Also

Adding Hierarchy Levels to a Selection Tree, to add levels to an existing Tree.

Selection Trees, for more information about using Selection Trees.

Using Selection Lists and Trees, for details on Selection element symbols.

Selection List, for information on creating non-hierarchical selection options.

Creating a Selection Tree from Selection Tree Component

To create a Selection Tree from individual columns, follow the steps below:

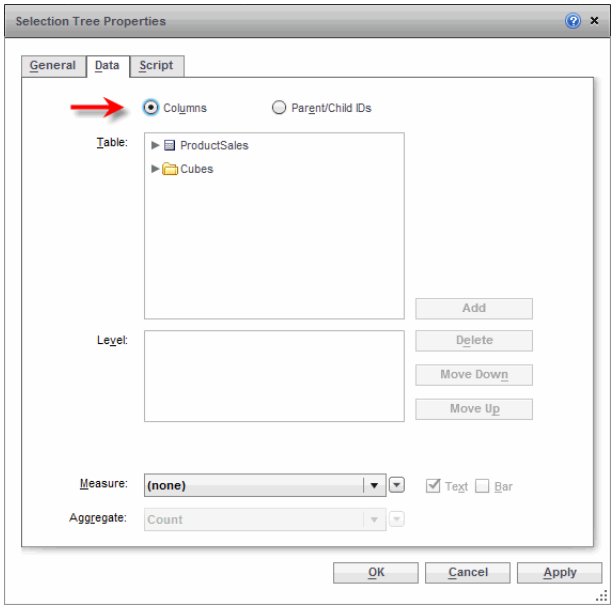
1. Drag a Selection Tree component from the Component tree onto the Viewsheet grid. This creates an empty Selection Tree element.

You can also drag-and-drop columns directly onto a Selection Tree element. Each new column you add is placed at the innermost level of the hierarchy.

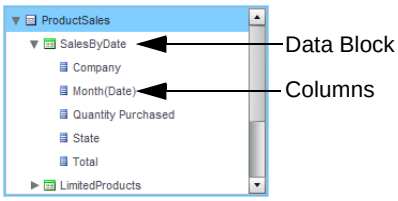
2. Right-click the Selection Tree, and select ‘Properties’ from the context menu. This opens the ‘Selection Tree Properties’ dialog box.

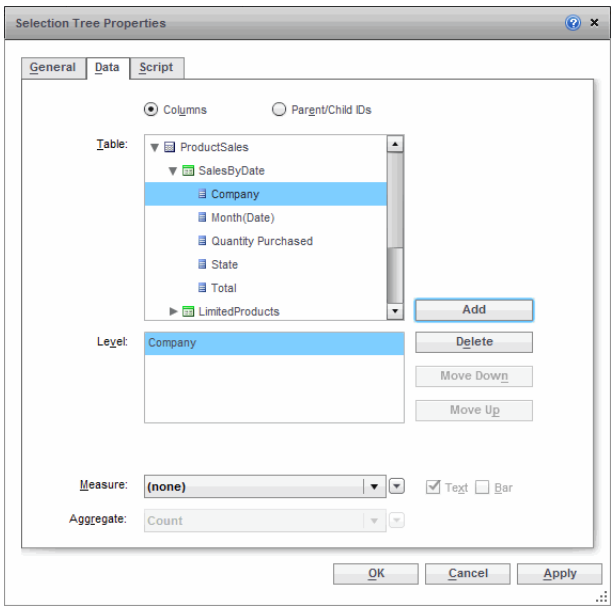
3. Select the **Data** tab.

4. At the top of the dialog box, select the ‘Columns’ option.



5. In the 'Table' menu, expand the desired Data Block to reveal its columns, or expand a cube dimension to reveal its levels.



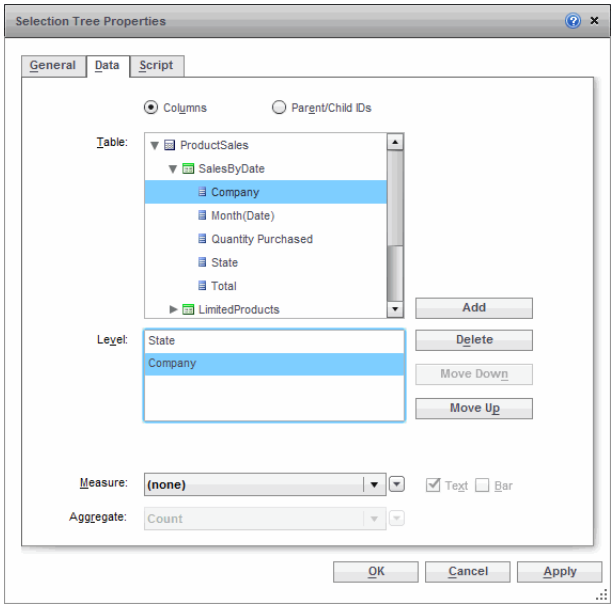


6. Select the column or level that you want to be the outer-most level of the Selection Tree hierarchy.

To add all columns or levels from a Data Block or cube dimension, select the desired Data Block or dimension, and click ‘Add’.

7. Click ‘Add’. This adds the column or level to the ‘Level’ list.

8. Repeat to add all desired levels of the hierarchy. To change the hierarchy ordering, use the ‘Move Down’ and ‘Move Up’ buttons. The order from top to bottom represents the nesting order of the Selection Tree, from outer to inner.



9. Click ‘OK’ to close the dialog box.

The Selection Tree is now populated based on the selected columns or dimension levels. The inner levels of the hierarchy show only the unique items compatible with corresponding outer level data. For example, the companies nested within the outer-level ‘CA’ node are only those companies whose corresponding state is CA.



See Also

Creating a Selection Tree from Recursive Hierarchy, to create a Selection Tree using parent and child IDs.

Adding Hierarchy Levels to a Selection Tree, to add levels to an existing Tree.

Interacting and Collaborating, for information on using dashboards.

Using Selection Lists and Trees, for details on Selection element symbols.

Selection List, for information on creating non-hierarchical selection options.

Creating a Selection Tree from Chart Data Source

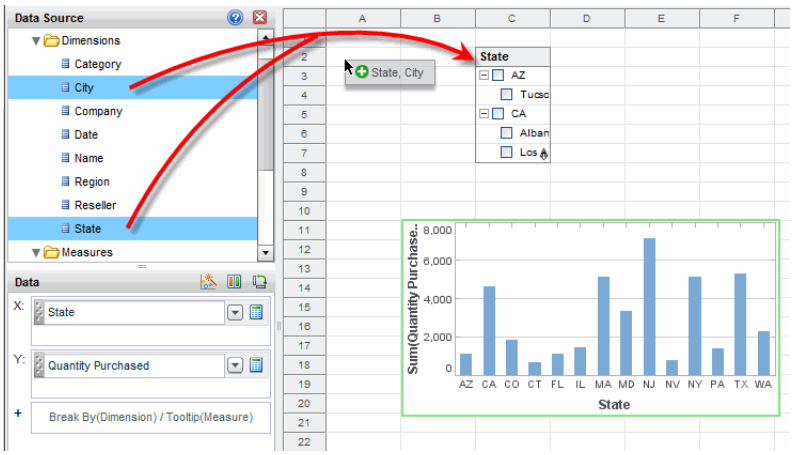
To create a Selection Tree from a chart Data Source tree, follow these steps:

1. Expand a 'Dimension' node in the Data Source tree. (This can belong to either a Data Block or cube.)

2. Select two or more columns from the 'Dimension' node, and drag them together onto the Viewsheet grid.

To include *all* levels of a cube dimension, simply drag the dimension node onto the Viewsheet.

This creates a Selection Tree based on the data in the selected fields.



See Also

Walkthrough: Interactive Chart Drilldown, in *Advanced Dashboard Scripting*, to adapt the X-axis binding based on user selection.

Creating Selection List from Chart Data Source, for information on the Selection List.

Creating a Chart, for information on using the Chart Editor.

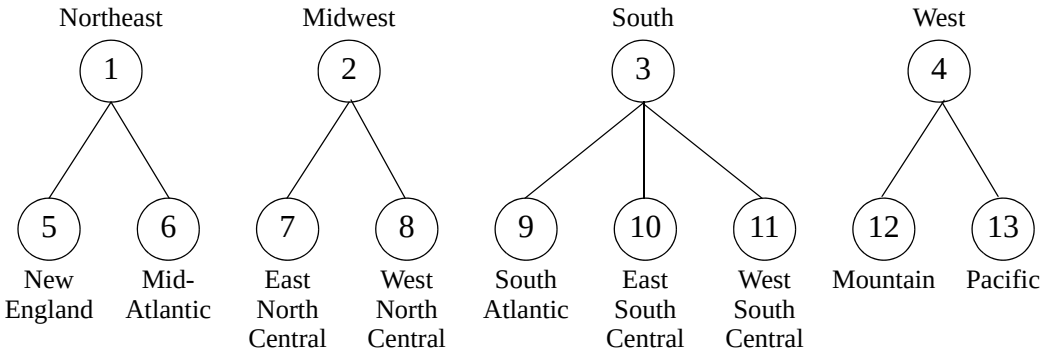
5.2.2 Creating a Selection Tree from Recursive Hierarchy

You can create a Selection Tree from recursive hierarchy information. For example, consider the following data. The "Child ID" is the ID of the node itself, while the "Parent ID" is the ID of the node's immediate parent. Nodes that lack a "Parent ID" comprise the top level of the tree.

PARENT ID	CHILD ID	LABEL
	1	Northeast
	2	Midwest
	3	South
	4	West

PARENT ID	CHILD ID	LABEL
1	5	New England
1	6	Mid-Atlantic
2	7	East North Central
2	8	West North Central
3	9	South Atlantic
3	10	East South Central
3	11	West South Central
4	12	Mountain
4	13	Pacific

The columns above represents the following tree structure.



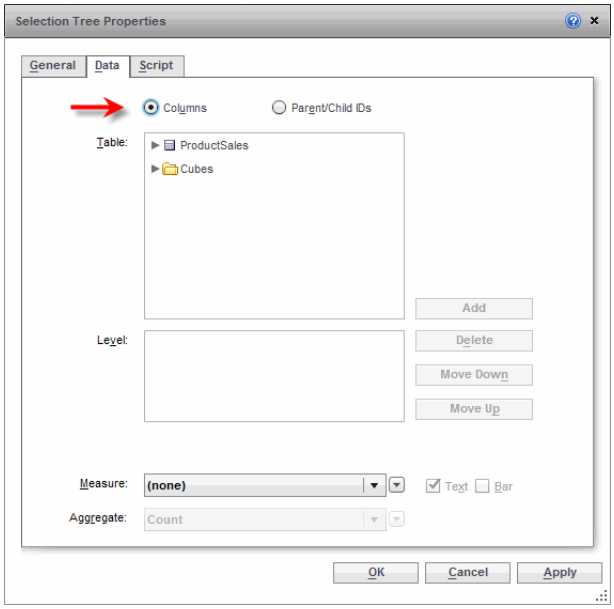
The “Parent ID” and “Child ID” columns indicate that nodes 5 and 6 share node 1 as parent, nodes 7 and 8 share node 2 as parent, nodes 9, 10, and 11 share node 3 as parent, and nodes 12 and 13 share node 4 as parent.

To create a Selection Tree from a recursive hierarchy defined by “Parent ID” and “Child ID” columns, follow the steps below:

1. Drag a Selection Tree component from the Component tree onto the Viewsheet grid. This creates an empty Selection Tree element.

2. Right-click the Selection Tree, and select ‘Properties’ from the context menu. This opens the ‘Selection Tree Properties’ dialog box.

3. Select the **Data** tab.



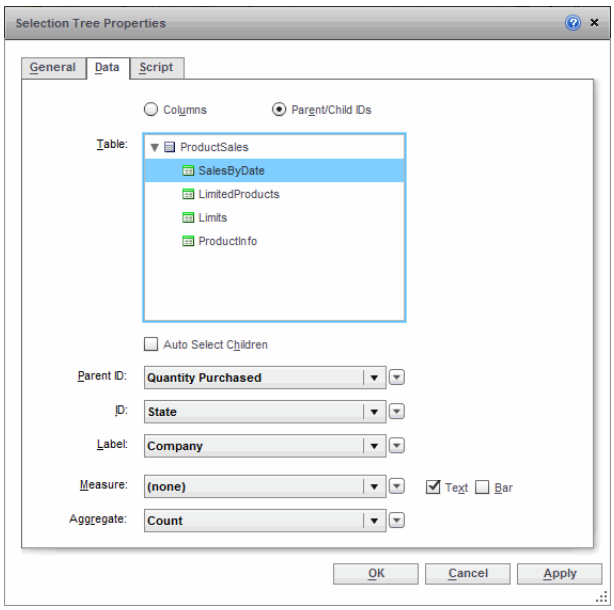
4. At the top of the dialog box, select the ‘Parent/Child IDs’ option.

5. In the ‘Table’ menu, expand the desired Data Block to reveal its columns.

6. In the ‘Parent ID’ field, select the column that contains the parent node IDs.

7. In the ‘ID’ field, select the column that contains the child node IDs.

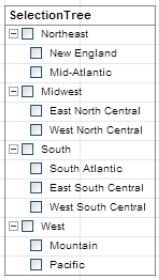
8. In the ‘Label’ column select the column that contains the labels corresponding to the child node IDs.



9. (Optional) Enable ‘Auto Select Children’ if you want the selection of a parent to automatically select the parent’s children.

10. Press ‘OK’ to close the dialog box.

The Selection Tree is now populated based on the selected “Parent ID” and “Child ID” columns.



See Also

Creating a Selection Tree from Individual Columns, to create a Selection Tree using nested columns.

Adding Hierarchy Levels to a Selection Tree, to add levels to an existing Tree.

Selection Trees, for information on how to use Selection Trees.

Using Selection Lists and Trees, for details on Selection element symbols.

Selection List, for information on creating non-hierarchical selection options.

5.2.3 Selection Tree Properties

The Selection Tree component’s ‘Properties’ dialog box provides **General** and **Data** tabs. The next sections discuss the component-specific properties available under these tabs.

See Also

Properties Dialog Box, for information about common component properties.

Format Dialog Box, for information on the ‘Format’ dialog box.

Selection Tree Properties: General Tab

The properties in the **General** tab in the ‘Selection Tree Properties’ dialog box are the same as those of the Selection List component.

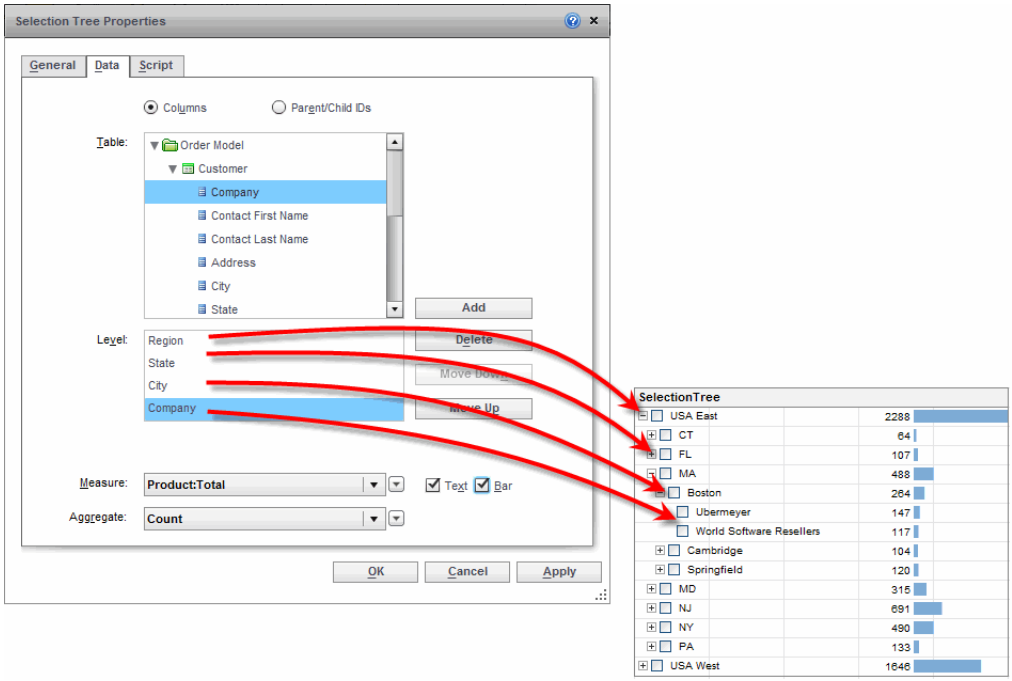
See Also

Selection List Properties: General Tab, for information on the **General** tab.

Selection Tree Properties: Data Tab

The **Data** tab in the ‘Selection Tree Properties’ dialog box provides the following Selection Tree-specific properties.

Table	The column or dimension that should be used to populate the Selection Tree
Level	The order of dimension nesting in the Selection Tree. Outer hierarchy levels are listed above inner hierarchy levels. See *Adding Hierarchy Levels to a Selection Tree* for instructions on altering the Selection Tree hierarchy.
Columns	Create the Selection Tree by using column information. See *Creating a Selection Tree from Individual Columns*.
Parent/ Child IDs	Create the Selection Tree by using recursive hierarchy. See *Creating a Selection Tree from Recursive Hierarchy*.
Auto Select Children	For Selection Tree based on recursive hierarchy, specifies that user selection of a parent also selects all the parent’s children.
Parent ID	For Selection Tree based on recursive hierarchy, column that contains the parent node IDs.
ID	For Selection Tree based on recursive hierarchy, column that contains the child node IDs.
Label	For Selection Tree based on recursive hierarchy, column that contains the labels corresponding to the child node IDs.
Measure	The field to aggregate and display next to the Selection Tree values.
Aggregate	The method of aggregation to be used for the ‘Measure’ field.
Text	Specifies that the aggregated measure is displayed as plain text. See *Displaying Aggregate Data in Selection List* for more information.
Bar	Specifies that the aggregate measure is displayed graphically as a proportional bar. See *Displaying Aggregate Data in Selection List* for more information.



5.2.4 Adding Hierarchy Levels to a Selection Tree

There are two ways to add additional hierarchy levels to a Selection Tree. The following sections explain both methods.

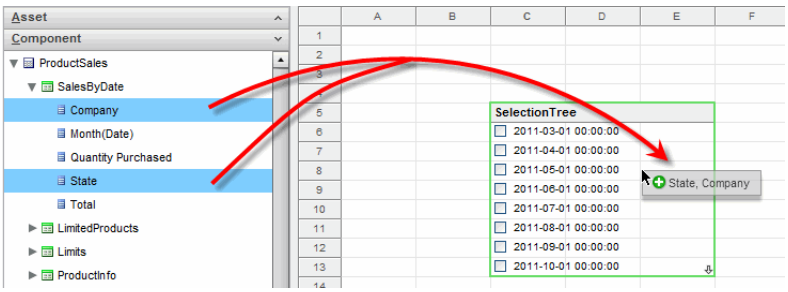
Adding Hierarchy Levels from Component Tree

To add levels to an existing Selection Tree hierarchy, follow these steps:

1. Expand a Data Block (table) or cube dimension in the Component tree.

2. Ctrl-click or Shift-click to select one or more columns in the Data Block or one or more levels in the cube dimension.

3. Drag the selected columns or levels onto an existing Selection Tree that is bound to the same Data Block or cube dimension. (Drop the columns or levels onto the Selection Tree when the green highlight appears.)



This adds the selected fields to the bottom (inner-most levels) of the Selection Tree hierarchy. The nesting order for the new levels is determined by the order in which you selected the fields in the Component tree.

Adding Hierarchy Levels from Properties Dialog Box

To add hierarchy levels to a Selection Tree, follow these steps:

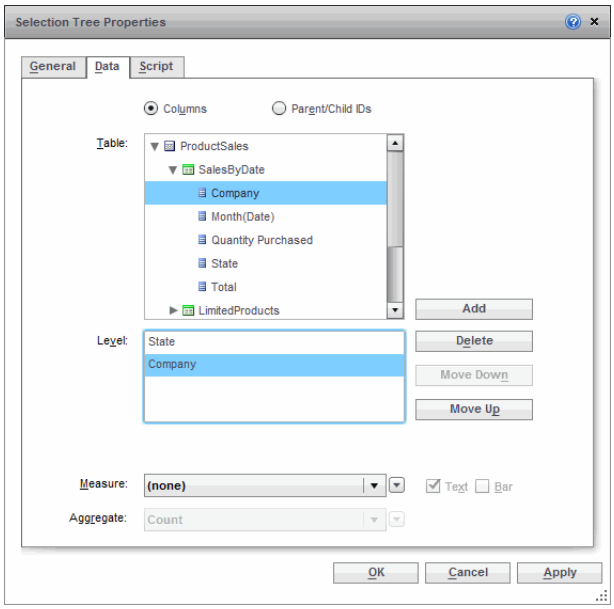
You can also drag and drop a single column directly onto the empty Selection List element.

1. Right-click on the Selection Tree, and select ‘Properties’ from the context menu. This opens the ‘Selection Tree Properties’ dialog box.

2. Select the **Data** tab.

3. From the ‘Table’ list, select the Data Block column or cube dimension that you wish to add to the hierarchy.

4. Click ‘Add’.



5. Repeat to add any additional levels to the hierarchy. To change the hierarchy ordering, use the ‘Move Down’ and ‘Move Up’ buttons. The order from top to bottom represents the nesting order from outer to inner.

6. Click ‘OK’ to close the dialog.

5.3 Selection Container

The Selection Container component serves two purposes:

- A Selection Container can display all of the selections (filters) that are currently operating on the Viewsheet. See *Displaying Selections*.
- A Selection Container can hold Selection List and Range Slider components. This allows end-users to add new selection components to a Viewsheet. See *Containing Selection Components*.

You can use a single Selection Container for both purposes.

See Also
Selection List, for a description of Selection Lists (discrete selections).
Range Slider, for information on Range Sliders (continuous selections).

5.3.1 Displaying Selections

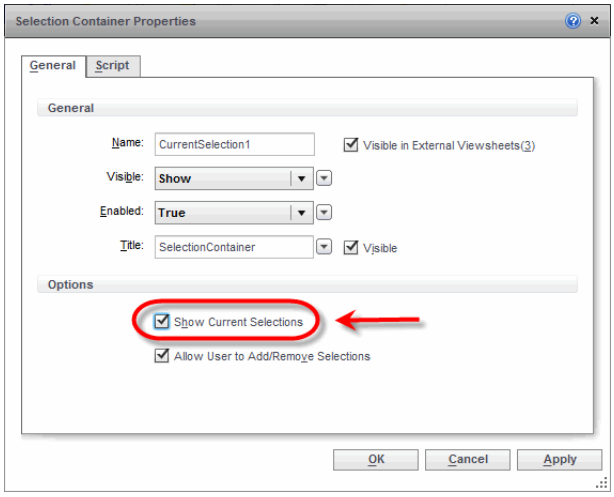
A Selection Container can display all of the selections (filters) that are currently specified in the Viewsheet. These include selections in Selection List, Selection Tree, Range Slider, or Calendar components.

To display all current selections within a Selection Container, follow the steps below:

1. Drag a Selection Container component from the Component tree onto the Viewsheet grid. This creates an empty Selection Container element.

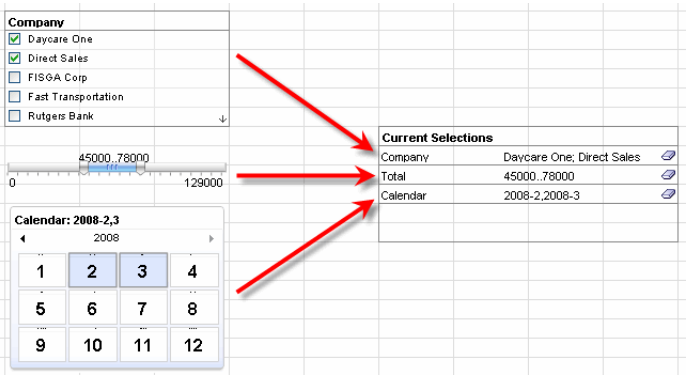
2. Right-click on the Selection Container, and select ‘Properties’ from the context menu. This opens the ‘Selection Container Properties’ dialog box.

3. In the ‘Options’ panel, enable ‘Show Current Selections’.



4. Click ‘OK’.

The Selection Container will now display the currently selected values for any Selection List, Selection Tree, Range Slider, or Calendar components in the Viewsheet.



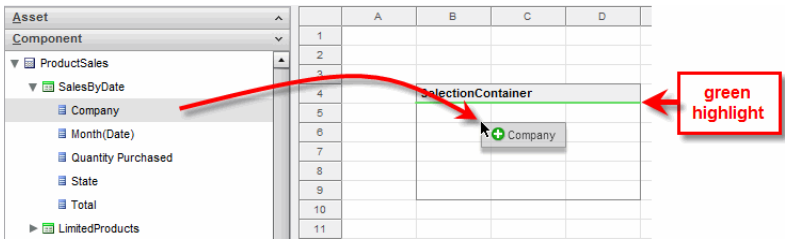
5.3.2 Containing Selection Components

A Selection Container can hold Selection List and Range Slider components. To add selection components to a Selection Container, follow the steps below:

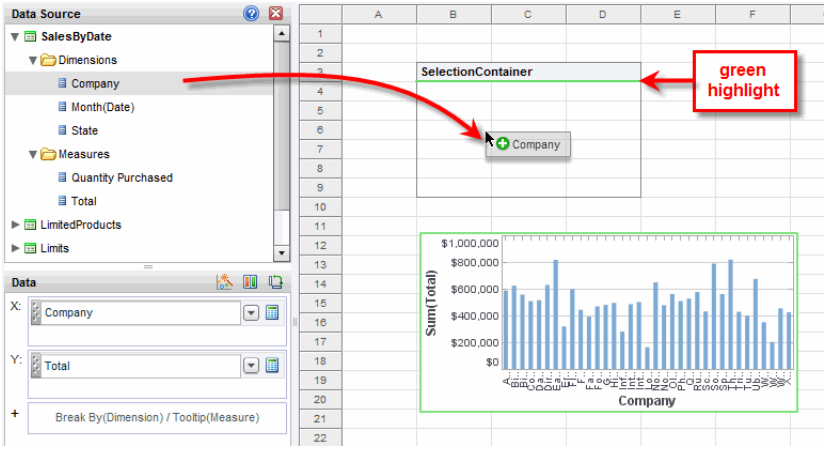
1. Drag a Selection Container component from the Component tree onto the Viewsheet grid. This creates an empty Selection Container element.

2. Drag a data field onto the Selection Container. (Use the green highlight to place the field as desired.) You can obtain this field from two sources:

   a. Drag a field from the Data Block in the Component tree onto the Selection Container.



   b. Drag a field from the Chart Editor onto the Selection Container.



This adds the chosen field to the Selection Container as a Selection List (categorical field) or a Range Slider (numerical or date field).

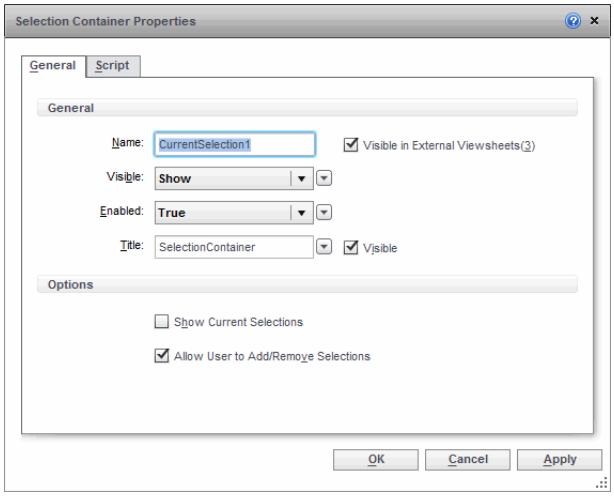
3. (Optional) Right-click on the Selection Container, and select 'Properties' from the context menu. This opens the 'Selection Container Properties' dialog box.

4. (Optional) To permit the end-user to drag new fields from the Chart Editor to the Selection Container, enable 'Allow New Selections in Viewer'.

5.3.3 Selection Container Properties

The 'Selection Container Properties' dialog box provides only the **General** tab. It offers the following unique properties for the Selection Container.

Show Current Selections	Enables the Selection Container to display the current state of all Viewsheet selection controls.
Allow New Selections in Viewer	Enables the end-user to drag new fields from the Chart Editor to the Selection Container.



See Also

Properties Dialog Box, for information about common component properties.

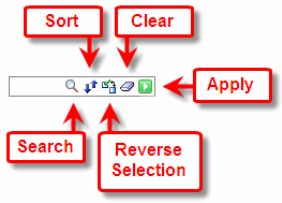
General Tab, for **General** tab properties that are common among components.

5.4 Using Selection Lists and Trees

Selection Lists and Selection Trees are simultaneously both input elements and output elements. When two or more Selection Lists or trees are linked to the same Data Block, query, or data model, the inputs to one of the lists or trees will modify the display in the other trees. See *Interacting with a Selection List* for an example.

This section explains how to make and clear selections, and explains the meaning of the Selection List symbols.

Figure 10. Selection List Toolbar



See Also

Selection Lists and *Selection Trees*, for information on how to use these components.

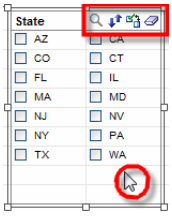
5.4.1 Submitting, Clearing, and Reversing Selections

The following sections explain how to submit, clear, and reverse a selection.

Submitting a Selection

To submit a selection, follow these steps:

1. Check the desired options in the Selection List or tree.
2. Hover the mouse over the list or tree to make the top-right toolbar visible.



3. Click the ‘Apply’ button.

When ‘Submit on Change’ is enabled, you can Ctrl-click to make multiple simultaneous selections.

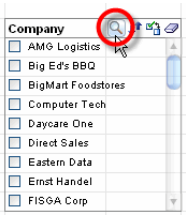
If the ‘Submit on Change’ option is set in the ‘Properties’ dialog box (enabled by default, see *Selection List Properties: General Tab*), you do not need to click the ‘Apply’ button. The selection is submitted each time you select or deselect one of the options.

Filtering Selections

When a Selection List or Selection Tree contains a large number choices, you may find it helpful to restrict or filter the available choices. This reduces the amount of list scrolling required, and saves time.

To filter a Selection List or Tree, follow the steps below:

1. Click the ‘Search’ button in the title bar of the Selection List. This opens a search box.



2. In the search box, type a portion of the choice that you wish to select.

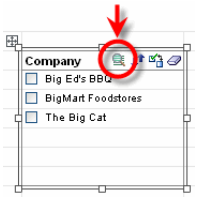


   As you type, the Selection List restricts the choices to that which match (at any location) the text you have entered.

3. When you have sufficiently restricted the available choices, make the desired selections by clicking the boxes. Note that the clear and

reverse functions will only apply to the filtered choices, and all values not displayed in the filtered list will be unaffected.

To exit the search box and restore all Selection List choices, click the 'X' button at the right side of the box. When the Selection List is currently filtered by a search term, the 'Search' button displays a unique icon in the title bar, shown below. Click it to modify or remove the search filter.

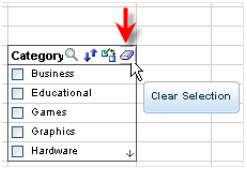


Clearing a Selection

To clear a selection, follow these steps:

1. Hover the mouse over the list or tree to make the top-right toolbar visible.

2. Click the 'Clear Selection' button.



This deselects all currently selected options.

Reversing a Selection

The Selection List offers a 'Reverse Selection' toolbar option. This inverts the current selection, selecting unchecked items, and deselecting checked items.

5.4.2 Selection List Symbols

The table below explains the five icons used in Selection Lists:

	**Data selected and included**. The user has explicitly selected this item, and the corresponding records are included in the returned data set.
	**Data selected but not included**. The user has explicitly selected this item, but more recent explicit selections in other Selection Lists have now made this selection incompatible. The corresponding data is therefore *not* included in the returned data set.

	**Data included but not selected**. The user has not explicitly selected this item, but has made other selections that *implicitly* select this item as well. The corresponding records are included in the returned data set (as a result of those other selections).
	**Data compatible but not included**. The user has not explicitly selected this item, and the item remains *compatible* with existing user selections. This item can therefore be selected to add additional records to the returned data set.
	**Data incompatible**. The user has not explicitly selected this item, and this item is *incompatible* with existing selections. (It appears listed under 'Others'). Selecting this *clears* all other selections and starts fresh.

5.4.3 Interacting with a Selection List

Walkthrough

Selection Lists allow you to gain tremendous insight into a data set. This example illustrates how you can use Selection Lists simultaneously for both input and output to offer end-users rapid exploratory capability.

1. Create a new Viewsheet based on the 'ProductSales' Worksheet.

2. Expand the 'SalesByDate' Data Block.

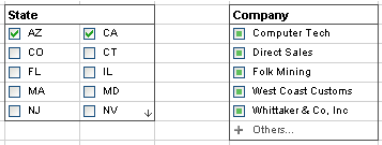
3. Drag the 'State' field from the Data Block onto the Viewsheet grid. This creates a new Selection List called 'State'.

4. Drag the 'Company' field from the Data Block onto the Viewsheet grid. This creates a new Selection List called 'Company'.

Make sure both lists are initially cleared.

5. Select 'AZ' and 'CA' from the 'State' Selection List.

   Note that the 'Company' Selection List has now automatically marked certain companies using green boxes. These are the companies that are compatible with your state selections (i.e., the companies residing within AZ and CA). Other companies are grouped together under the heading 'Others'.



6. Expand the 'Others' node, and select 'Eastern Data'.

   Note that this clears your previous selections in the 'State' Selection List, because 'Eastern Data' is not compatible with either 'AZ'

and ‘CA’. The ‘State’ Selection List now automatically shows ‘NY’ with a green box, indicating that this state is compatible with the selection of ‘Eastern Data’.

5.5 Range Slider

The Range Slider element filters data to fall within a specified range. It can be used on both Date fields and Numeric fields, and provides the ability to filter data by consecutive dates or numbers.

5.5.1 Creating a Range Slider

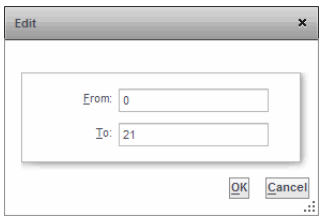
To create a Range Slider, follow the steps below:

1. Drag a Range Slider element from the Component tree into the Viewsheet grid.

2. Drag a Data Block column (from Repository Tree or Viewsheet Table) the onto the Range Slider element. This binds the Range Slider to the selected column.

3. Adjust the selection range by dragging the slider end-handles. (You can also click on an end-handle and press the arrow keys on the keyboard.)

4. To enter precise values for the start-point or end-point, right-click on the Range Slider and select ‘Edit’ from the context menu. In the ‘Edit’ dialog box, enter the desired values and press ‘OK’.



5. To quickly set the right side of the range to its maximum, double-click on the right end-handle. To quickly set the left side of the range to its minimum, double-click on the left end-handle. To quickly set the entire range to its full extent, double-click on the slider body.

You can also bind the Range Slider to a field by using the **Data** tab of the ‘Range Slider Properties’ dialog box. This approach also allows you to filter aggregated data displayed in Chart and Crosstab components. See *Range Slider Properties: Data Tab* for more information.

See Also

Range Slider Properties: Data Tab, for information about the **Data** tab.

Example:
Range Slider

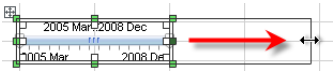
In this example you will use a Range Slider to filter sales based on both date and quantity. Follow the steps below:

1. Create a new Viewsheet based on the 'ProductSales' Worksheet.
2. Drag a 'Range Slider' element from the Component tree into the Viewsheet grid.

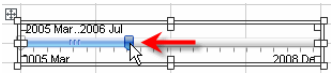
3. Expand the 'SalesByDate' Data Block, select the 'Date' field, and drag it onto the Range Slider element.



4. Adjust the size of slider unit by dragging the outer handles.



5. Adjust the selection range by dragging the slider endpoints.



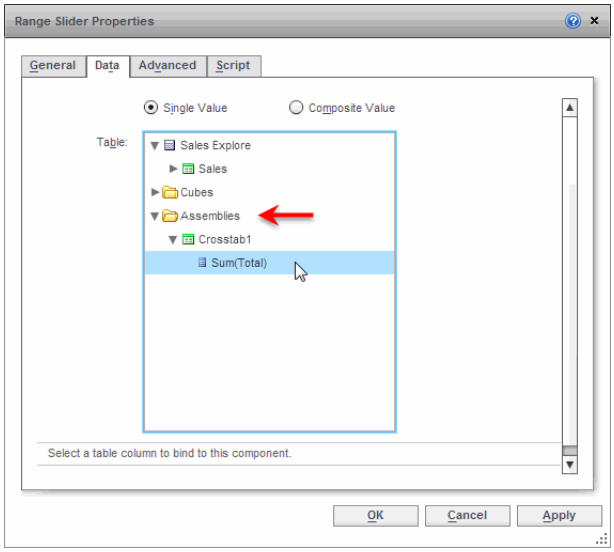
¢

5.5.2 Filtering Post-Aggregate Data with a Range Slider

You can use a Range Slider to filter the post-aggregate data displayed by a Chart, Crosstab, or Freehand Table. To do this, follow the steps below:

1. Drag a 'Range Slider' element from the Component tree into the Viewsheet grid.

2. Right-click on the Range Slider and select 'Properties' from the context menu. This opens the 'Range Slider Properties' dialog box.
3. Select the **Data** tab. In the 'Table' panel, expand the 'Assemblies' node, and select the component and aggregated field that you want to filter.



4. Press ‘OK’ to close the dialog box.

This Range Slider will now filter the post-aggregate values in the selected Chart, Crosstab, or Freehand Table.

See Also
Filtering a Table and *Filtering a Chart* in *Interacting and Collaborating*, to filter data without adding a separate Range Slider.

5.5.3 Range Slider Properties

The Range Slider component’s ‘Properties’ dialog box provides **General**, **Data**, and **Advanced** tabs. The next sections discuss the component-specific properties available under these tabs.

See Also
Properties Dialog Box, for information about common component properties.
Format Dialog Box, for information on the ‘Format’ dialog box.

Range Slider Properties: General Tab

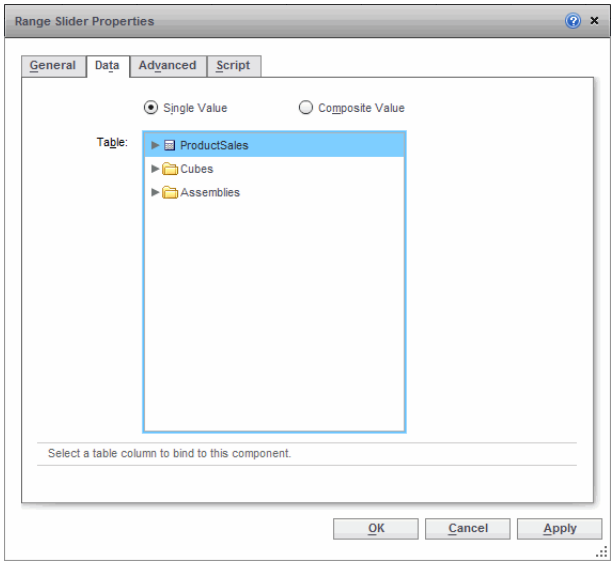
The **General** tab in the ‘Range Slider Properties’ dialog box provides only the common properties.

See Also
General Tab, for information about common properties in the **General** tab.

Range Slider Properties: Data Tab

The **Data** tab in the 'Range Slider Properties' dialog box provides the following Range Slider-specific properties:

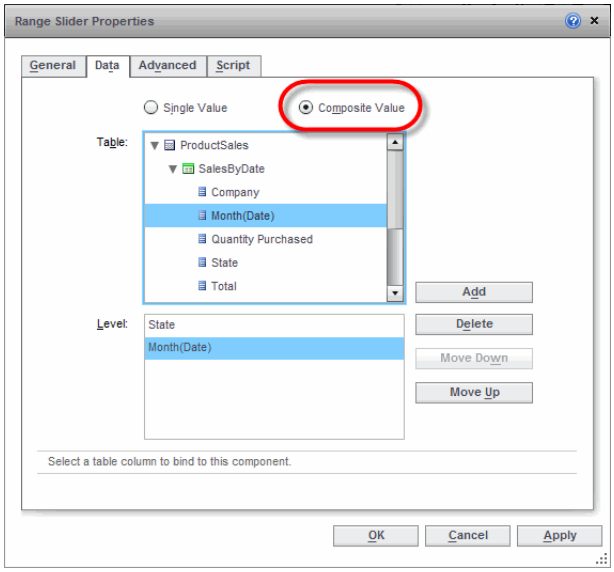
Single Value	Filter data based on one field from the Data Block (default). Only numeric and date type fields are supported.
Composite Values	Filter data based on the composition of two or more fields from the Data Block. See *Composite Value* for more information.



Composite Value

The 'Composite Value' option allows you to filter data using the composite of two or more columns of any type. All data types are supported. To set the composite value option, follow the steps below.

1. Right-click on the Range Slider, and select 'Properties' from the context menu.
2. Under the **Data** tab, select 'Composite Value'.
3. Select two or more columns to form the composite values for filtering. Click OK.



Example: Range Slider Composite Value

As an example, consider the following Data Block:

YEAR	MONTH
2000	Jan
2000	Feb
2008	Mar
2004	Mar
2006	Feb

Using the composite value of these two fields, the Range Slider values are the following tuples:

```
{(2000,Jan),(2000,Feb),(2004,Mar),(2006,Feb)(2008,Mar)}
```

Selecting the composite value `(2000,Jan)`, for example, filters the data set based on the constraint (Year=2000, Month=Jan).

¢

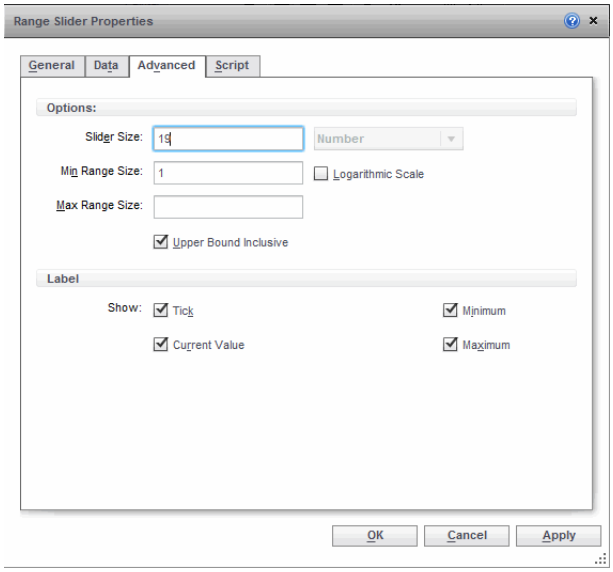
Range Slider Properties: Advanced Tab

To create a coarser range selection (e.g., date based on "quarter of year" or string based on "first letter",) create the appropriate groupings using in the Data Worksheet.

The **Advanced** tab in the 'Range Slider Properties' dialog box provides the following Range Slider-specific properties.

Slider Size	The range of data that is displayed at a given time. There are three cases: **String**: For string-type fields, the 'Slider Size' specifies the number of "periods," i.e., the number of consecutive (alphabetically ordered) values that should be simultaneously selected. **Numeric**: For numeric-type fields, the 'Slider Size' specifies the number of 'Min Range Size' periods spanned by the slider. The total numerical range simultaneously selected is therefore 'Slider Size' * 'Min Range Size'. **Date**: For *date-time* type fields, 'Slider Size' specifies the number of periods to be simultaneously selected, where the period duration is given by the adjacent popup menu: 'Year', 'Month', 'Day', 'Hour', or 'Minute'. For *date* type fields, the options are 'Year', 'Month', and 'Day'. For *time* type fields, the options are 'Hour' and 'Minute'. Note: For 'Day', 'Hour', and 'Minute' settings, if the number of ticks exceeds 1500, the range size is automatically adjusted to yield a smaller number of ticks.
Min Range Size	For numeric-type fields, this is the smallest amount by which the range can be adjusted.
Max Range Size	Specifies the largest range that the user can specify. The units are the same as the 'Slider Size'.
Upper Bound Inclusive	Specifies that the upper bound of the selected range is included in the selection. (For example, if the user sets the slider upper limit to "10", the result set includes data with values of "10" for this field).
Logarithmic Scale	Sets logarithmic scaling for the slider, which is useful for traversing large numerical ranges. The logarithm base is selected automatically.
Label	The labels to display on the Range Slider: **Tick**: Show the increments by which the slider can be adjusted. **Current Value**: Permanently display the slider value above the slider. **Minimum**: Show the minimum slider value at the left of the slider. **Maximum**: Show the maximum slider value at the right of the slider.

The 'Slider Size' can also be adjusted graphically in the Viewsheet by dragging the slider endpoints. See *Creating a Range Slider* for an example.



See Also

Grouping, in *Advanced Data Mashup*, for information in creating groups.

5.6 Calendar

A Calendar allows you to select data by time range, measured in weeks or months. It also facilitates comparative aggregation, for example, the side-by-side comparison of total quantity purchased during the months of Dec. 2005 and Dec. 2006.

See Also

Calendars, for information on how to use a Calendar.

5.6.1 Creating a Calendar

To create a calendar, follow the steps below:

1. Drag a 'Calendar' element from the Component tree onto the Viewsheet grid. This creates a new Calendar element.



2. Drag a date column from a Data Block (on the tree) and drop it onto the calendar. This links the calendar to the data represented by the selected date field.

3. Set properties of the Calendar using the ‘Calendar Properties’ dialog box (see *Calendar Properties*) or by using the mini-toolbar (see *Calendars*).

5.6.2 Calendar Properties

The Calendar component’s ‘Properties’ dialog box provides **General**, **Data**, and **Advanced** tabs. The next sections discuss the component-specific properties available under these tabs.

See Also
Calendars, for information on using the Calendar component.
Properties Dialog Box, for information about common component properties.
Format Dialog Box, for information on the ‘Format’ dialog box.

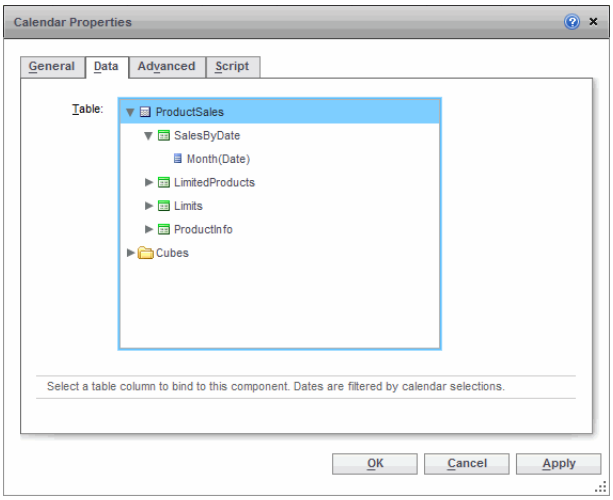
Calendar Properties: General Tab

The **General** tab in the ‘Calendar Properties’ dialog box provides only the common general properties.

See Also
General Tab, for information about common properties in the **General** tab.

Calendar Properties: Data Tab

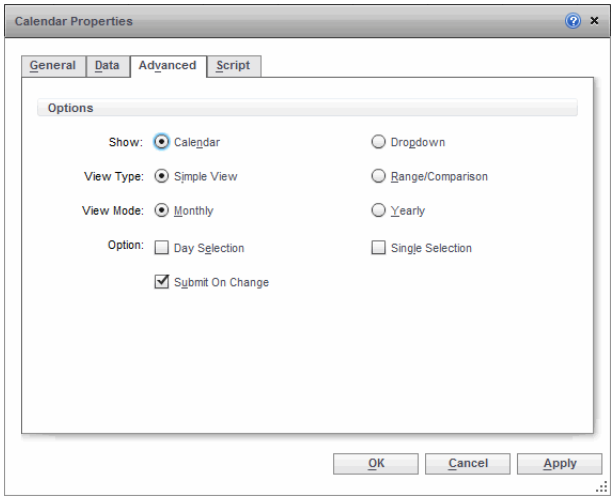
The **Data** tab in the ‘Calendar Properties’ dialog box allows you to choose the date field on which Calendar selections operate. Only columns with date formatting are displayed.



Calendar Properties: Advanced Tab

The **Advanced** tab in the 'Calendar Properties' dialog box provides the following Calendar-specific properties

OPTION	DESCRIPTION
'Show'	**Calendar**: Displays the calendar as a traditional wall calendar. **Dropdown**: Displays the calendar as a drop-down list.
'View Type'	**Simple View**: Displays a single calendar unit. **Range/Comparison**: Displays two calendars side by side, facilitating date comparison.
'View Mode'	**Yearly**: Displays all twelve months in a single calendar. **Monthly**: Displays a single month. **Day Selection**: Enables selection of individual days on the calendar. **Single Selection**: Prevents the user from selecting multiple date units on the Calendar. (This property is not supported for Viewsheets displayed using HTML, such as those on iPad.)



6 Output Components

Graphical output components display summarized data using a familiar measurement device (Gauge, Cylinder, Thermometer, Sliding Scale, etc.). The highly graphical nature of these objects makes them ideal candidates for presenting key indicators. Like data views, graphical output components are controlled by selection and input components.

6.1 Thermometer/Cylinder/Sliding Scale/Gauge

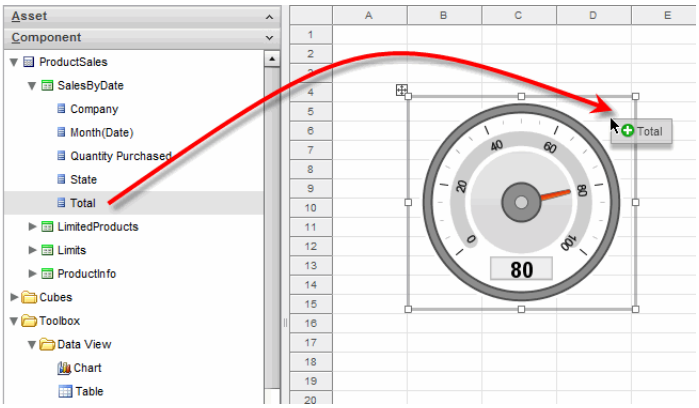
Thermometers, Cylinders, Scales, and Gauges are identical in terms of data binding and functionality. They display a summarized value for a single column in a Worksheet Data Block, query, or model. These output components differ only in appearance.

6.1.1 Creating an Output Component

To add an output component to a Viewsheet, follow the steps below:

1. Drag the desired component from the Component tree into the grid.

2. Drag a column from the Data Block (at the top of the Components tree), and drop it onto the output component.



This binds the output component to the Data Block column, so that the component displays a summary of the values in that column.

3. To specify the summarization method, follow these steps:

   a. Right-click the component, and select ‘Properties’. This opens the ‘Properties’ dialog box for the component.

   b. In the **Data** tab of the ‘Properties’ dialog box, select a summarization method from the ‘Aggregate’ menu, and click ‘OK’.

6.1.2 Output Component Properties

The ‘Properties’ dialog box for output components provides the following tabs: **General**, **Data**, and **Advanced**. The next sections discuss the output-specific properties available under these tabs.

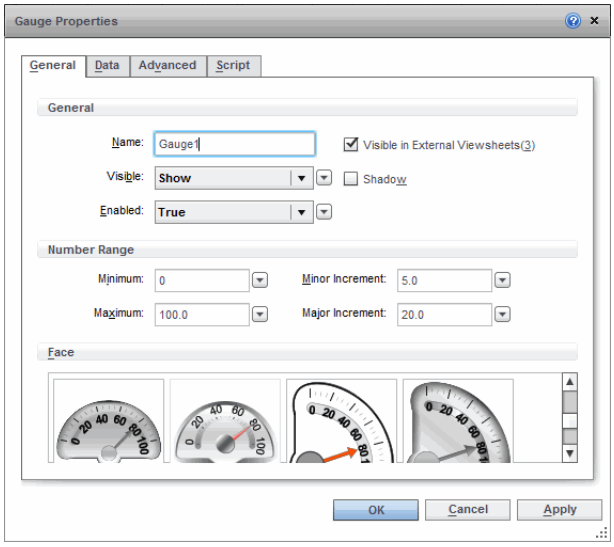
See Also

Properties Dialog Box, for information about common component properties.

Output Component Properties: General Tab

The **General** tab in the ‘Properties’ dialog box provides the following unique properties for output elements.

Number Range	**Minimum**: The starting value on the display. **Maximum**: The ending value on the display. **Minor Increment**: The minimum resolution on the display. **Major Increment**: The points at which the values are displayed. If you leave the Number Range properties blank, the Gauge will automatically create a scale appropriate to the data.
Face	The visual form of the element. Click on the desired skin to select it.
Shadow	Adds a drop shadow to the component.



Note: If you are connecting to the server via “https” in Internet Explorer, you may need to change the following Internet Explorer setting in order for the ‘Face’ property to update correctly: In Internet Explorer, open the ‘Internet Options’ dialog box, and click the ‘Browsing History Settings’ button to open the ‘Temporary Internet Files’ dialog box. Select ‘Every time I visit the web page’ under ‘Check for newer versions of stored pages’.

See Also

General Tab, for **General** tab properties that are common among components.

Output Component Properties: Data Tab

The **Data** tab in the 'Properties' dialog box for output components provides only the common data properties.

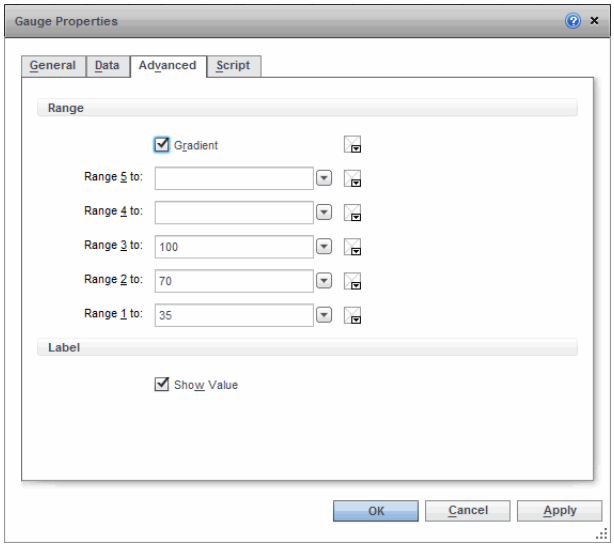
See Also

Data Tab, for information about common properties in the **Data** tab.

Output Component Properties: Advanced Tab

The **Advanced** tab in the 'Properties' dialog box for output components provides the following properties:

Range	A set of colored zones to display on the output component. See *Setting Ranges for Output Components*.
Show Value	Provides a digital display of the current value shown on a Gauge.



Setting Ranges for Output Components

To set multiple ranges for an output component, follow the steps below:

1. Open the **Advanced** tab of the Properties dialog box. In the 'Range' panel you will see three editable drop-down lists.

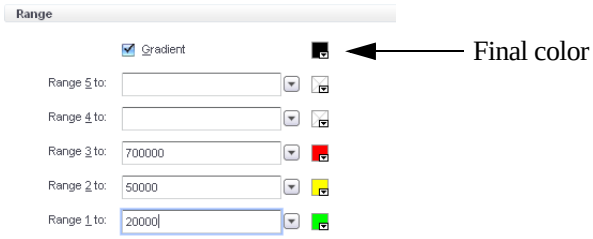
2. Enter a value for 'Range 1 to' and choose a color from the pop-up menu. This sets the region from the minimum value on the scale to the specified value.

3. Enter a value into ‘Range 2 to’ and choose a color from the pop-up menu. This sets the region from the end of ‘Range 1’ to the specified value.

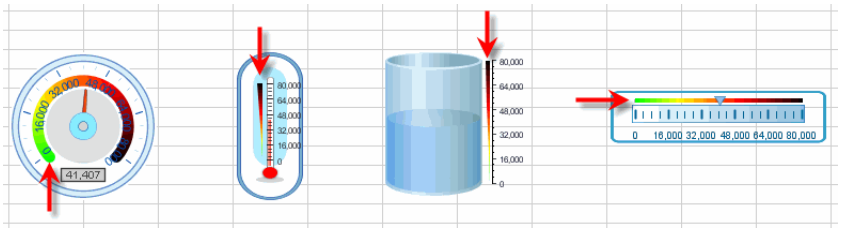
4. Repeat the above steps to add any additional thresholds.

5. (Optional) Select ‘Gradient’ to smoothly blend the zone colors, if desired.

6. (Optional) If you have selected ‘Gradient’, choose a color for the final blend.



The placement of the scale differs for each output component, as shown below.



6.2 Text

Text elements can be used to display static labels as well as summarized data for a single column in a Worksheet Data Block, query, or data model.

6.2.1 Creating a Text Element

To add a text into a Viewsheet, drag a Text component from the **Component** panel into the Viewsheet grid. Use the mouse to move and resize the element.

Double-click inside the Text element to edit the text. The text that you enter will automatically wrap to fit within the component boundaries. If you set the ‘Auto Size’ property (see *Text Properties: General Tab*), the Text component will expand vertically to display the entirety of the text that you enter.

To bind a text element to a data column, drag the desired column from the Data Block (at the top of the **Component** panel) and drop it onto the element.

6.2.2 Text Properties

The Text component's 'Properties' dialog box provides the following tabs: **General** and **Data**. The next sections discuss the text-specific properties available under these tabs.

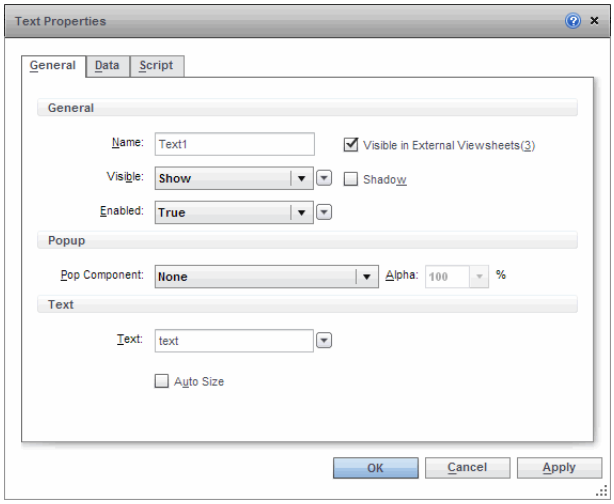
See Also

Properties Dialog Box, for information about common component properties.

Text Properties: General Tab

The **General** tab in the 'Text Properties' dialog box provides the following text-specific property.

Text	Display a label, or specify a single calculated value.
Shadow	Adds a drop shadow to the component. (Drop shadow for a Text component is not preserved on export.)
Pop Component	Select an individual component or *Container* to display as a pop-up interface. When the user clicks the text, the selected component is displayed at the location of the text. Otherwise, the selected component is not visible in the Viewsheet.
Alpha	Sets the transparency of the pop-up component or group. An 'Alpha' value of 100% indicates complete opacity. An 'Alpha' value of less that 100% allows underlying components to be partially visible through the popped-up component or group.
Auto Size	Allows the bottom border of the Text component to expand vertically so that all of the text in the component is visible.



See Also

General Tab, for information about common properties in the **General** tab.
Advanced Dashboard Scripting, to replace the fixed text with a formula.

Text Properties: Data Tab

The **Data** tab in the 'Text Properties' dialog box provides only the common data properties.

See Also

Data Tab, for information about common properties in the **Data** tab.

6.2.3 Adding Highlights to Text

To add a highlight to a text element, right-click the text element, and select 'Highlight' from the context menu. This opens the 'Highlight' dialog box.

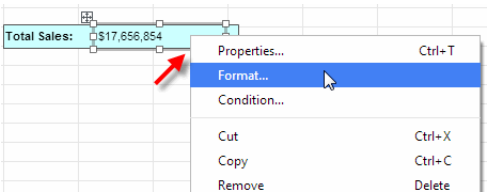
See Also

Highlights, for instructions on how to use the 'Highlight' dialog box.

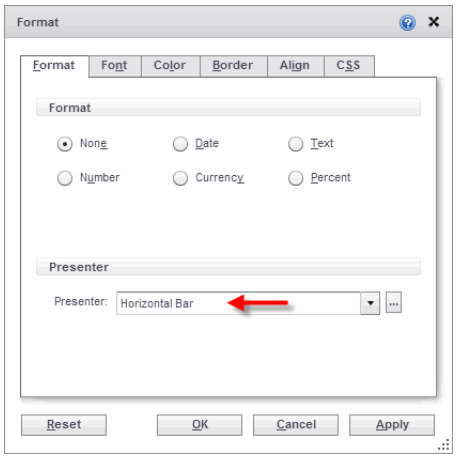
6.2.4 Displaying Data in a Text Component Graphically

To display a numerical value in a Text component using a graphical representation such as a colored bar or QR code, follow the steps below.

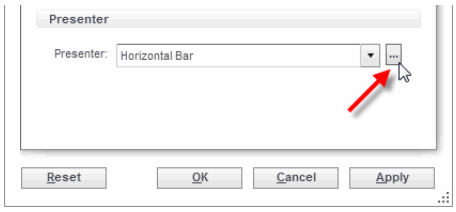
1. Right-click on a Text component that displays a numerical value, and select 'Format' from the context menu.



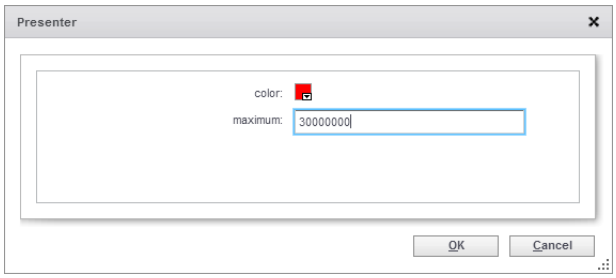
2. On the **Format** tab of the 'Format' dialog box, select a desired graphical representation from the 'Presenter' menu.



3. Press the ‘Options’ button to open the ‘Presenter’ dialog box.



4. Specify the desired values for the presenter options, and press ‘OK’.



See *Presenter* for more information about the options.

5. Press ‘OK’ to close the ‘Format’ dialog box. This applies the presenter to the Text component.



See Also

Displaying Table Cell Values Graphically, to use a presenter in a table.
Presenter, for more information about presenters.

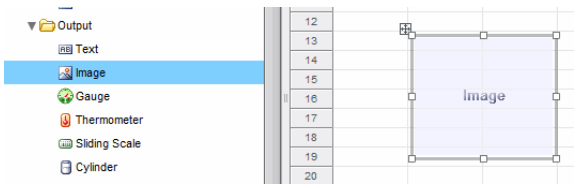
6.3 Image

An Image element displays an image file (gif, jpg, png) on the Viewsheet.

6.3.1 Creating an Image

To add an Image element to a Viewsheet, follow the steps below:

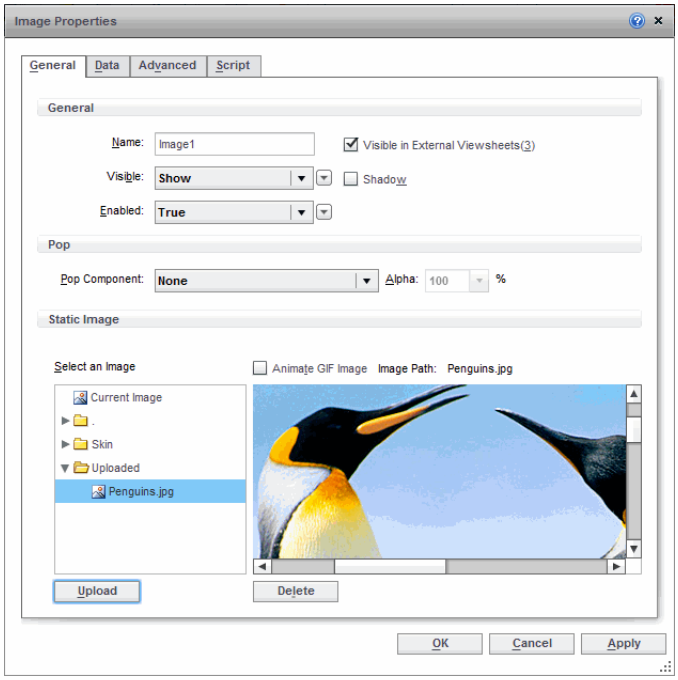
1. Drag an Image element from the Component tree onto the Viewsheet grid. Use the handles to move and resize the element.



2. Right-click the image element, and select ‘Properties’. This opens the ‘Image Properties’ dialog box. Select the **General** tab.

Uploaded image files are stored in the asset.dat file together with the Viewsheet.

3. To upload a local image file, click the ‘Upload’ button at the bottom of the dialog box, and choose the file. The uploaded file is now listed under the ‘Uploaded’ folder.



To specify an image that already exists on the server (in a directory on the classpath), enter the resource path (e.g. ‘/library/image.gif’) as an expression in the ‘Dynamic Image Selection’ field, under the **Advanced** tab.

4. Select the desired image from the ‘Uploaded’ folder, and click ‘OK’.

The image is added to the Viewsheet. By default, the image appears at its full size. To scale the image to the Image container, select ‘Scale Image’ from the **Advanced** tab.

See Also

Scaling an Image, for information on scaling an image in the Viewsheet.
Creating a Shape, for information on creating primitive shapes.
Changing Images with Script, in *Advanced Dashboard Scripting*, for information on changing image content using script.

6.3.2 Image Properties

The Image component's 'Properties' dialog box provides the following tabs: **General**, **Data**, and **Advanced**. The next sections discuss the image-specific properties available under these tabs.

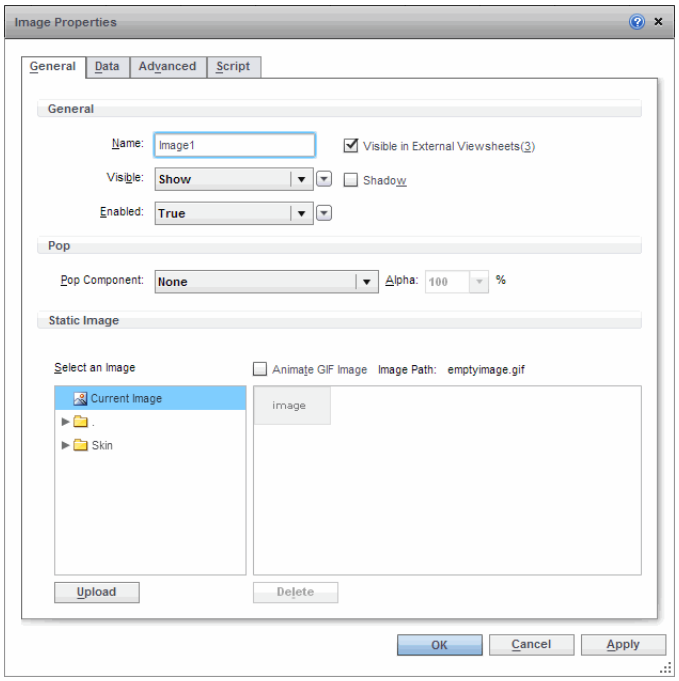
See Also

Properties Dialog Box, for information about common component properties.

Image Properties: General Tab

The **General** tab in the 'Image Properties' dialog box provides the following image-specific property.

Pop Component	Select an individual component or *Container* to display as a pop-up interface. When the user clicks the image, the selected component is displayed at the location of the image. Otherwise, the selected component is not visible in the Viewsheet.
Alpha	Sets the transparency of the pop-up component or group. An 'Alpha' value of 100% indicates complete opacity. An 'Alpha' value of less that 100% allows underlying components to be partially visible through the popped-up component or group.
Static Image	Select an existing server image, or 'Upload' an image to the server. The 'Skins' folder provides a few predefined images which are useful as backgrounds. The 'Theme Background' has the special property of automatically adapting to match the prevailing color scheme of the User Portal.
Shadow	Adds a drop shadow to the component.



See Also

General Tab, for information about common properties in the **General** tab.
Changing Images with Script, in *Advanced Dashboard Scripting*, for information on specifying an image file through scripting.
Creating an Image, for instructions on how to upload an image.

Image Properties: Data Tab

The **Data** tab in the 'Image Properties' dialog box provides the common data properties. The 'Aggregate' value that you specify here is used as the value for the Image highlight condition.

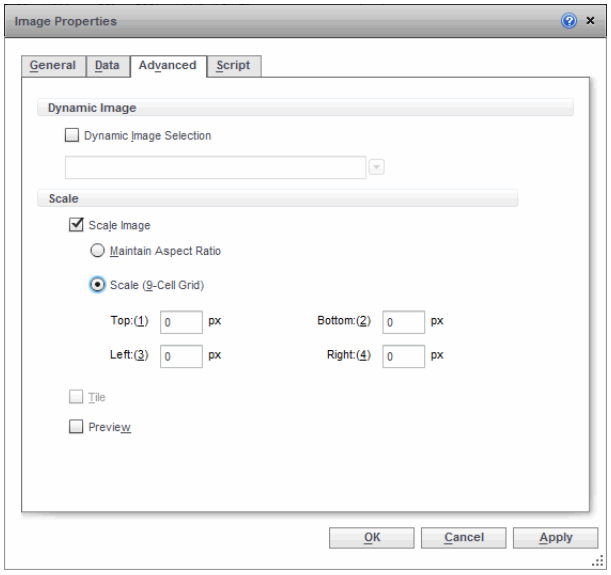
See Also

Adding Highlights to an Image, for information on conditional formatting.
Data Tab, for information on common properties in the **Data** tab.

Image Properties: Advanced Tab

The **Advanced** tab in the 'Image Properties' dialog box provides the following image-specific properties.

Dynamic Image Selection	Specifies a method for dynamically setting the image. See *Changing Images with Script*, in *Advanced Dashboard Scripting*, for more information.
Scale Image	When selected, specifies that images should be scaled to the containing component boundary. The following options are available: **Maintain Aspect Ratio**: The image's width to height ratio is preserved when the image is resized, preventing the image from being distorted. **Scale**: The image is divided into nine regions, which scale independently. See *Scaling an Image*.
Tile	Tiles (repeats) the image horizontally and vertically within the component boundaries.
Preview	Displays a thumbnail of the image as it will appear with the applied scaling.



6.3.3 Scaling an Image

To scale an image, you need to enable image scaling, and then choose the method by which images should scale. Follow the steps below:

1. Right-click the Image component, and select ‘Properties’ from the context menu. This opens the ‘Image Properties’ dialog box.

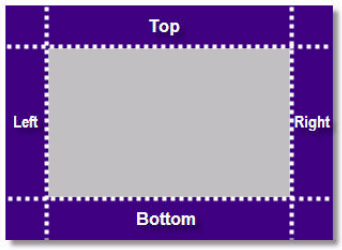
2. Click the **Advanced** tab, and enable the ‘Scale Image’ option.

3. To keep image proportions constant during scaling, select ‘Maintain Aspect Ratio’. This allows the image to scale without distortion.

4. To selectively scale only parts of the image (e.g., the middle of the image), select ‘Scale (9-Cell Grid)’.

   This divides the image into a 3x3 grid of cells, shown below. The ‘Top’ and ‘Bottom’ cells are scaled only horizontally. The ‘Left’ and ‘Right’ cells are scaled only vertically. The corner cells are not scaled, and maintain their original size. The middle (gray) cell is scaled in both horizontal and vertical directions.

Tip: For an image with a border, you may want to keep the border width constant as the image is scaled. To do this, set the pixel values slightly greater than the border thickness.



5. If you have selected ‘Scale (9-Cell Grid)’, enter the desired sizes of the ‘Top’, ‘Bottom’, ‘Left’, and ‘Right’ regions in pixels.

6. Click ‘OK’ to close the dialog box.

The image will now be scaled in the specified manner when you adjust the Image boundaries.

See Also
Creating a Background, to use an image as Viewsheet or component background.

6.3.4 Adding Highlights to an Image

To add a highlight to an image element, right-click the image element, and select ‘Highlight’ from the context menu. This opens the ‘Highlight’ dialog box.

See Also
Highlights, for instructions on how to use the ‘Highlight’ dialog box.

6.3.5 Tinting an Image

To add colored tint to an image, follow the steps below:

1. Right-click the image, and select ‘Format’ from the context menu. This opens the ‘Format’ dialog box.

2. Select the **Color** tab.

3. In the ‘Text’ panel, select a color from the menu.

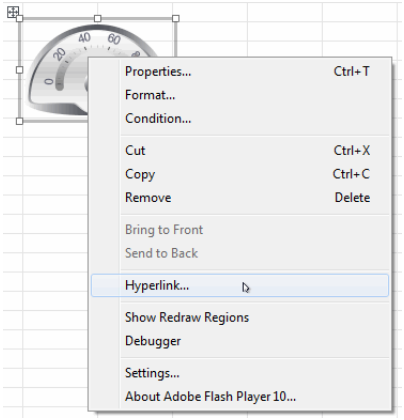
4. Click ‘OK’ to close the dialog box.

The color you selected is now used to tint the image.

See Also
Color Tab, for more information about the options on the **Color** tab.

6.4 Adding Hyperlinks to Output Components

In output components, hyperlinks can be attached to the component body (e.g., the entire Gauge). To create hyperlinks on an output component, right-click the component and select ‘Hyperlink’ from the context menu. This opens the ‘Hyperlink’ dialog box.



See Also

Hyperlinks, for information on the ‘Hyperlink’ dialog box.

7 Input Components

Note: The Viewsheet's *onRefresh Handler* script is re-executed each time the user makes a selection from an Input Component.

You can enter data into Viewsheets using the input components provided in the toolbox (Slider, Spinner, Combo Box, etc.) Input elements, unlike selection elements, do not merely filter a fixed set of data. Instead, they cause re-execution of queries and modify the original result set.

See Also
Output Components, for information on graphical presentation of data.
Selection Components, for information on filtering the data view display.
Data View Components, for information on tables and charts.

7.1 Using Input Components

Input components enable the user to enter values into a Viewsheet. There are three ways that you can use the entered input values:

Note: You can also add a Worksheet Embedded Table as a Viewsheet component, so that users can directly enter values. See *Creating an Embedded Table*.

- Pass the input values to an Embedded Table in the linked Worksheet: *Embedded Tables* in a Data Worksheet are often linked (via joins, etc.) to other Data Blocks in such a way that changing values in the Embedded Table alters the data contained in those other blocks. This allows the user to perform *what-if* analysis, manipulating the values in the Embedded Table to see how a hypothetical scenario would impact their real data.

Note: An input component that is not linked to a Variable can still be referenced in Worksheet script as a parameter having the name of the component.

- Pass the input values to a Variable in the linked Worksheet: *Variables* in a Data Worksheet are used in Data Block conditions, which effectively filter the data contained in a given Data Block.
- Pass the input values to set component properties or conditions in the Viewsheet.

Note: The CheckBox control does not support the Embedded Table interface.

The following sections discusses the Embedded Table and Variable interfaces. Viewsheet input components (Slider, Radio Button, etc.) provide a way to alter the values set within the Worksheet's Embedded Table or Variable without accessing the Worksheet directly. This allows you to explore the same data by making intuitive control adjustments, rather than by typing in numbers.

For information on using inputs to dynamically adjust component properties and conditions, see *Dynamic Properties*.

See Also
Creating an Embedded Table, in *Advanced Data Mashup*, for basic concepts.

Creating a Variable, in *Advanced Data Mashup*, for information on Variables.

7.1.1 Configuring an Embedded Table

To pass inputs from a Viewsheet to the underlying Worksheet, the Worksheet must contain an Embedded Table. Follow the steps below to add and configure the Embedded Table in the Worksheet.

1. In the Worksheet, click the 'New Object' button in the toolbar, and select the 'Embedded Table' option. The pointer changes to a crosshair.

Any cell in the Embedded Table can receive its value from a Viewsheet control.

2. Drag the mouse across the Worksheet grid to create the new Embedded Table. Create as many rows and columns as needed for the application.

3. Right-click the Embedded Table, and select 'Properties' from the context menu. This opens the 'Properties' dialog box.

4. In the 'Name' field, enter a name for the Embedded Table. This is the name by which you will identify the table in the Viewsheet.

5. Click 'OK' to close the 'Properties' dialog box.

6. Edit the column names, if desired. (Meaningful column names make it easier to bind the Viewsheet input control.)

7. Enter default values into the cells of the Embedded Table.

   A cell that is linked to a Viewsheet control will be dynamically updated to the value set in the control. A cell that is not linked to a Viewsheet control will retain the fixed value you enter.

The Embedded Table is usually connected to other Data Blocks by one or more joins.

8. Connect the Embedded Table to other Data Blocks in the Worksheet, as desired.

See Also

Linking a Viewsheet to a Data Source, for an application example.
Creating an Embedded Table, in *Advanced Data Mashup*, for basic concepts.
Joining Tables, in *Advanced Data Mashup*, for information on creating joins.
Renaming a Column, in *Advanced Data Mashup*, to set column names.

7.1.2 Passing Inputs to an Embedded Table or Variable

To link a Viewsheet input control (other than a Check Box) to an Embedded Table or Variable in the Worksheet, follow these steps:

1. Right-click on the input control, and select ‘Properties’.

2. Select the **Data** tab.

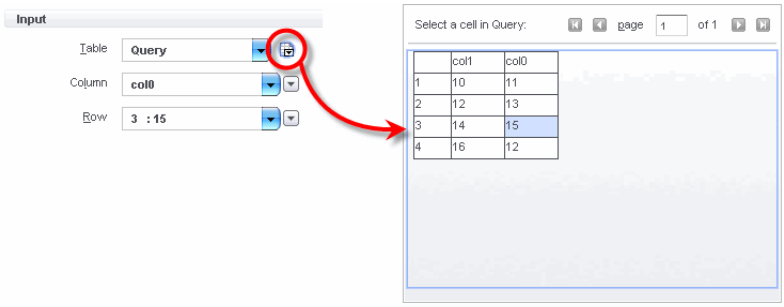
Multiple controls can be linked to the same Embedded Table.

3. Click the ‘Target’ menu:

   a. To access an Embedded Table, expand the ‘Table’ node, and select the desired Embedded Table Data Block of the Worksheet.

   b. To access a Variable, expand the ‘Variable’ node, and select the desired Variable asset in the Worksheet. Skip to the final step below.

4. For an Embedded Table, click the ‘Browse’ button next to the ‘Table’ field. This displays a representation of the Embedded Table.



5. In the displayed Embedded Table, click to select the table cell into which input values should be passed. This automatically populates the ‘Column’ and ‘Row’ menus with the corresponding cell indices.

   Alternatively, specify the target cell directly from the provided menus:

The ‘Row’ menu displays the format “row: row value.” For example, “1:25” indicates that input values are passed to the second row of the Embedded Table, which currently contains the value “25.”

   a. From the ‘Column’ menu, select the appropriate field of the Embedded Table Data Block.

   b. From the ‘Row’ menu, select the row of the table to which the input element should provide data.

6. Click ‘OK’ to exit the ‘Properties’ dialog box.

To set the options displayed by the Input control, see *Creating a Radio Button/Check Box/Combo Box*.

The control is now linked to the Worksheet’s Embedded Table or Variable, and changes you make using the control will alter the values in the specified Embedded Table cells or Variable.

See Also

Input Components, for information on controls for Embedded Table input.
Creating an Embedded Table, for directly entering data into an Embedded Table.
Creating an Embedded Table, in *Advanced Data Mashup*, for basic concepts.
Creating a Variable, in *Advanced Data Mashup*, for basic concepts.
What-If Analysis, in *Advanced Data Mashup*, for an application example.
Dynamic Properties, to control properties from input elements.

7.2 Slider and Spinner

Sliders and Spinners are interactive input elements that can supply numerical data to Embedded Tables and Variables in the underlying Worksheet and Viewsheet component properties. Sliders and Spinners differ from one another only in appearance.

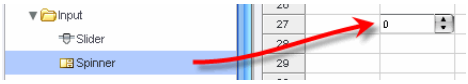
See Also

Passing Inputs to an Embedded Table or Variable, to manipulate the Worksheet.
Dynamic Properties, to manipulate Viewsheet properties.

7.2.1 Creating a Slider or Spinner

To create a Spinner, follow the steps below. (The Slider is identical.)

1. Drag the ‘Spinner’ element from the component tree into the Viewsheet grid. This creates an empty Spinner in the Viewsheet grid.



2. Right-click on the Spinner, and select ‘Properties’ from the context menu. This opens the ‘Spinner Properties’ dialog box.

3. Under the **General** tab, specify the ‘Minimum’ and ‘Maximum’ values.

4. If you will use the Spinner to control dynamic properties of another element, follow these steps:

   a. Enter a ‘Name’ for the control. This is the name by which the control is identified in dynamic property menus. (See *Dynamic Properties*.)

   b. Click ‘OK’ to finish.

Note: If you plan to use the Slider or Spinner to set dynamic properties, you do not need to specify an Embedded Table or Variable in the **Data** tab.

5. If you will use the Spinner to control values in a Worksheet Embedded Table or Variable, follow the steps below:

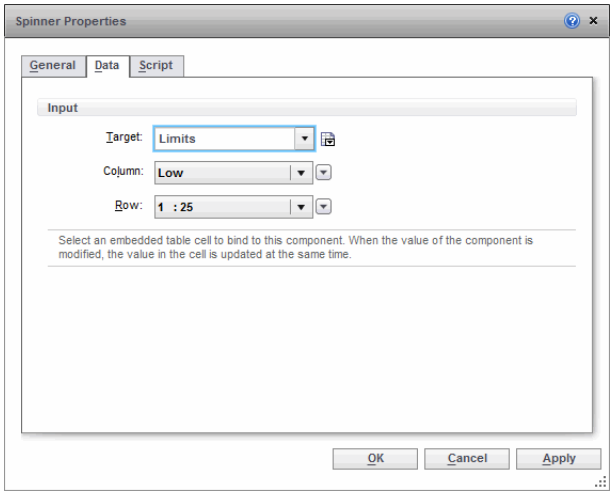
   a. Select the **Data** tab.

   b. In the 'Target' list, expand the 'Tables' or 'Variables' node, and select an Embedded Table or Variable from the linked Worksheet.

   c. For an Embedded Table, select the desired column from the 'Column' list and the desired row from the 'Row' list. The row and column together define the cell to which the Spinner provides data.

   d. Click 'OK'.

The format shown is "row: value." The left number is the row index, the right number is the current value in that row (i.e., the default value).



Example: Spinner Control of Embedded Table

In this example, you will use a Spinner control to display only values in a table that exceed the specified Spinner value.

1. Create a new Worksheet.

2. In the following steps, add a new Embedded Table containing a single numeric data cell:

   a. Click the 'New Object' button on the toolbar, and select 'Embedded Table'.

   b. Drag across the Worksheet grid to select three vertical cells, and release.

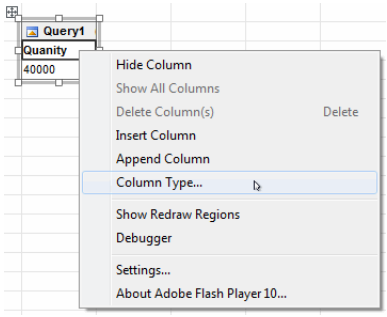


This creates a new Embedded Table named ‘Query1’, with just one data cell.

c. Double-click the column header cell (‘col0’), and change the name to “Quantity”.

d. In the data cell of the Embedded Table, enter the value “4000.”

e. Right-click the column header cell (‘Quantity’), and select ‘Column Type’ from the context menu.



This opens the ‘Column Type’ dialog box.

f. In the ‘Column Type’ dialog box, set ‘Type’ to ‘Integer’. Click ‘OK’.

3. Expand the ‘Orders’ data source on the Asset tree, and drag the ‘Sales by Category’ query onto the Worksheet. This creates a new table called ‘Sales by Category1’.

Note that a *Range Slider* provides a very simple way to filter based on value, without requiring Embedded Tables.

4. In the following steps, join the Embedded Table to the ‘Sales by Category1’ table. The tables will be joined on the ‘Qty Sold’ column using an inequality join condition:

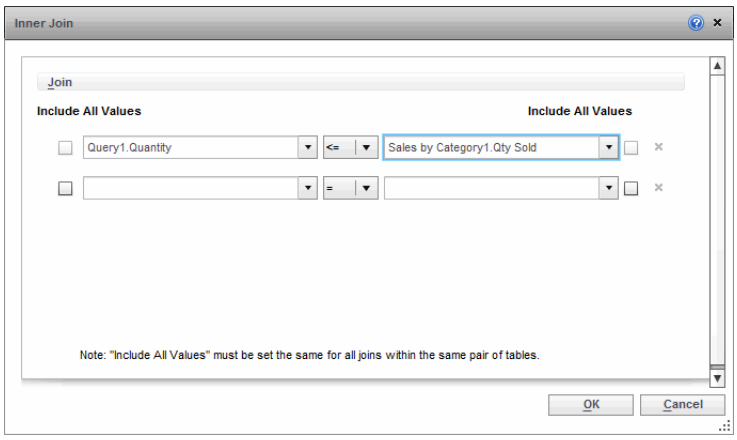
a. Select both of the tables (Shift-click).

b. Click the ‘Join Table’ button on the toolbar, and select ‘Inner Join’. This opens the ‘Inner Join’ dialog box.

c. On the left side, select ‘Query1.Quantity’ from the top menu.

d. On the right side, select ‘Sales by Category1.Qty Sold’ form the top menu.

e. From the center menu (comparison operator), select '<=', as shown below.



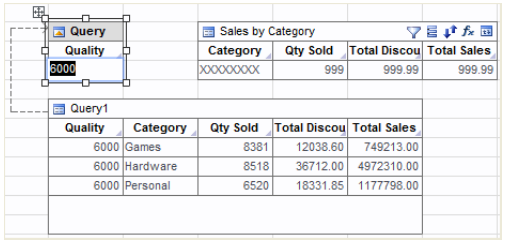
f. Press 'OK'.

The tables are now joined on the 'Qty Sold' column, creating a new table called 'Query2'. The values shown in the 'Qty Sold' column of joined table 'Query2' will only be those which equal or exceed the 'Quantity' value in the Embedded Table.

5. Verify that the joined table is filtered correctly:

a. Press the 'Change View' button in the toolbar of the 'Query2' table and select 'Live Preview'. Expand the table to show all the data. (Only values of 'Qty Sold' greater than or equal to 4000 are shown.)

b. Enter "6000" as the value in the Embedded Table, and observe how the data in table 'Query2' is filtered accordingly.

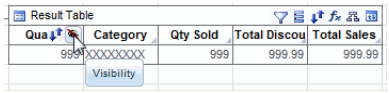


6. Prepare the joined table to be displayed in a Viewsheet:

a. Right-click the 'Query2' table title bar and select 'Properties' from the context menu. Change the name of the joined table to "Result Table".

b. Press the ‘Change View’ button in the toolbar, and select ‘Default’ to exit live-data mode.

c. In the left-most table column named ‘Quantity’, click the ‘Visibility’ button to hide the column. (This column shows the value from the Embedded Table, and does not need to be displayed in the Viewsheet.)



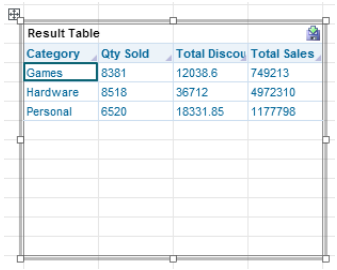
d. Save the Worksheet as “Large Quantity”.

7. In the following steps, create a new Viewsheet to display the ‘Result Table’ from the Worksheet:

a. Create a new Viewsheet based on the ‘Large Quantity’ Worksheet.

b. From the Component tree, drag the ‘Result Table’ Data Block onto the Viewsheet grid. This creates a table called ‘Result Table’.

c. Drag the table handles to make the table taller, so that about 10 rows are shown.



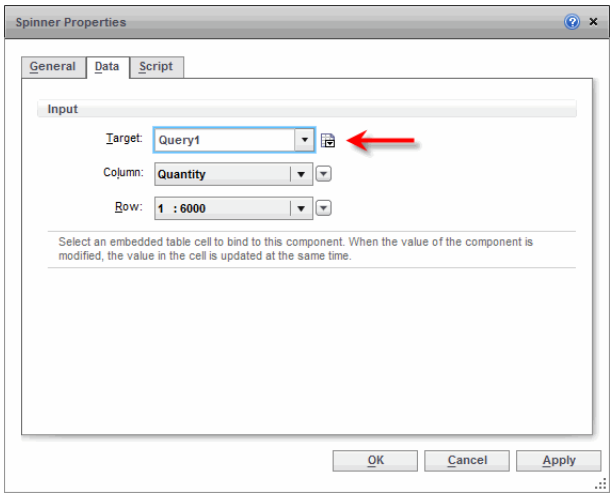
8. Add a Spinner element to control the value in the Embedded Table (thus filtering the displayed ‘Result Table’):

a. Drag a Spinner from the **Component** tree onto the Viewsheet grid.

b. Right-click the Spinner, and select ‘Properties’ from the context menu. This opens the ‘Spinner Properties’ dialog box.

The 'Row' menu shows a single row because the Embedded Table has only one row. The value shown is the current value of the Embedded Table cell, which was specified earlier.

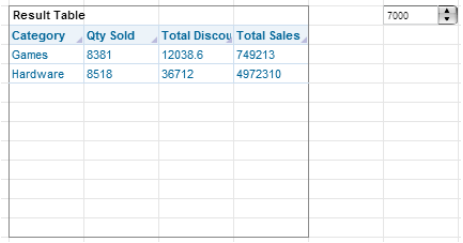
c. Select the **Data** tab. In the 'Target' menu, expand the 'Table' node, and select 'Query1'. (This is the ID of the Embedded Table in the Worksheet.)



d. Select the **General** tab. In the 'Number Range' panel, set 'Maximum' to "10000."

e. Press 'OK'.

9. Test the Spinner control by entering different values. Observe how the values in the table change: The table only displays records for which 'Qty Sold' is greater than the value you choose in the Spinner.



¢

See Also

Using Input Components, for details on the Embedded Table interface.
Creating a New Data Worksheet, in *Advanced Data Mashup*.
Creating a New Embedded Table, in *Advanced Data Mashup*.
Dynamic Properties, to control properties with input elements.

7.2.2 Slider and Spinner Properties

The ‘Properties’ dialog box for Slider and Spinner components provides the following tabs: **General**, **Data**, and **Advanced**. The next sections discuss the input-specific properties available under these tabs.

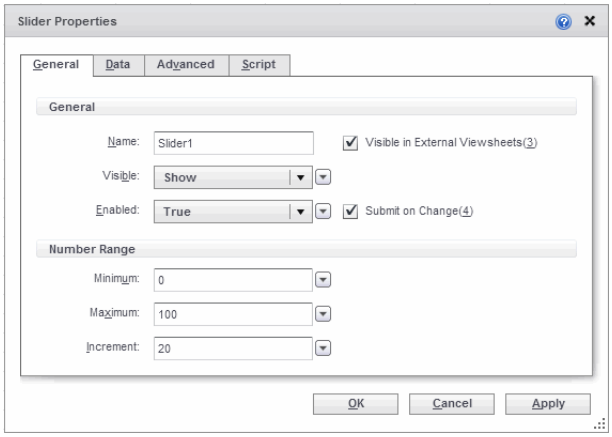
See Also

Properties Dialog Box, for information about common component properties.

Slider and Spinner Properties: General Tab

The **General** tab in the Slider and Spinner ‘Properties’ dialog box provides the following unique properties.

Submit on Change	When enabled (default), submits the selected value immediately upon selection. Otherwise, a separate ‘Submit’ button is required.
Number Range	**Minimum**: The starting value **Maximum**: The ending value **Increment**: The points at which the values are displayed



See Also

General Tab, for **General** tab properties that are common among components.

Slider and Spinner Properties: Data Tab

The **Data** tab in the ‘Properties’ dialog box for input components provides the following input-specific properties:

Target	An Embedded Table or Variable in the Worksheet. The ‘Browse’ button to the right allows you to interactively select the row and column of the Embedded Table.
Column	The column of the Worksheet Embedded Table into which the input value should be passed.

Row	The row of the Worksheet Embedded Table into which the input value should be passed. (The 'Row' menu displays the format "row: row value.")

See *Using Input Components* for details on setting and using these properties.

See Also
Data Tab, for **Data** tab properties that are common among components.

Slider and Spinner Properties: Advanced Tab

The **Advanced** tab in the 'Properties' dialog box for the Slider component provides the following specific property:

Label	The labels to display on the Slider: **Tick**: Show tick marks along the slider scale. **Current Value**: Display the current value above the slider when dragging. **Labels**: Show value labels along the slider scale. ('Tick' must be enabled.) **Minimum**: Show the minimum slider value at the left of the slider. (Enabled by default when 'Labels' is enabled.) **Maximum**: Show the maximum slider value at the right of the slider. (Enabled by default when 'Labels' is enabled.)

7.3 Radio Button, Check Box, Combo Box

Radio Buttons, Check Boxes, and Combo Boxes are used to enter numeric or non-numeric data into an Embedded Table or Variable in the underlying Worksheet. They can also be used to set dynamic properties of other Viewsheet components, e.g., visibility, color, etc.

Radio Buttons, Check Boxes, and Combo Boxes offer similar functionality. Radio Buttons and Combo Boxes allow only a single item to be selected, while Check Boxes allow multiple items to be selected.

See Also
Using Input Components, for details on Embedded Table and Variable interfaces.
Dynamic Properties, to control properties using input elements.
Creating a Slider or Spinner, for an Embedded Table usage example.

7.3.1 Creating a Radio Button/Check Box/Combo Box

To create a Radio Button/CheckBox/ComboBox, follow the steps below.

1. Drag a Radio Button/CheckBox/ComboBox control from the Component tree onto the Viewsheet grid.

2. Right-click the control and select ‘Properties’ from the context menu. This opens the Radio Button/CheckBox/ComboBox ‘Properties’ dialog box.

3. If you will use the Radio Button/CheckBox/ComboBox to set dynamic properties for another element, follow these steps:

   a. Select the **General** tab, and enter a ‘Name’ for the control. This is the name by which the control is identified in dynamic property menus. (See *Dynamic Properties*.)

   b. Click ‘OK’ to finish.

Note: If you plan to use the input control to set dynamic properties, you do not need to specify an Embedded Table under the **Data** tab.

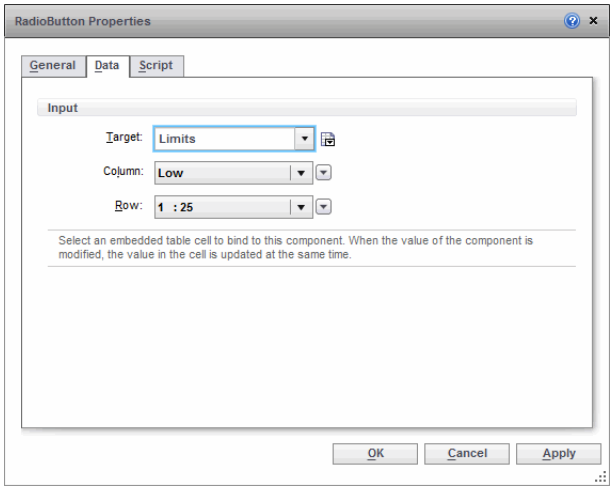
4. If you will use the Radio Button/CheckBox/ComboBox to set values in a Worksheet Embedded Table or Variable, follow the steps below. (See *Passing Inputs to an Embedded Table or Variable* for more details.)

   a. Select the **Data** tab.

   b. From the ‘Target’ list, select an Embedded Table Data Block or Variable in the linked Worksheet.

   c. For an Embedded Table, from the ‘Column’ list, select the desired field of the Embedded Table. From the ‘Row’ list, select the row of the table to which the input control should provide data.

The format shown is “row: value.” The left number is the row index, the right number is the current value in that row (i.e., the default value).



   d. Click ‘OK’.

5. Select the **General** tab, and populate the Radio Button/CheckBox/ ComboBox control with the values it will display. There are three methods:

   a. Manually enter a list of values and labels: See *Populating Label/ Value Pairs Manually*.

   b. Specify a query to generate a list of values and labels: See *Populating Label/Value Pairs from Query*.

   c. For ComboBox only, you can acquire the input via a calendar widget. See *Acquiring Date Input Using a Calendar Control*.

6. Click 'OK' to close the dialog box.

See Also

Using Input Components, for details on Embedded Table and Variable interfaces.
Dynamic Properties, to control properties from input elements.
Populating Label/Value Pairs Manually, to manually enter values/labels.
Populating Label/Value Pairs from Query, to obtain values/labels from query.

Populating Label/Value Pairs Manually

Applies to Radio Buttons, Check Boxes, and Combo Boxes.

To manually enter label/value pairs for a Radio Button, Check Box, or Combo Box, follow the steps below:

1. Right-click the input control, and select 'Properties' from the context menu. This opens the 'Properties' dialog box.

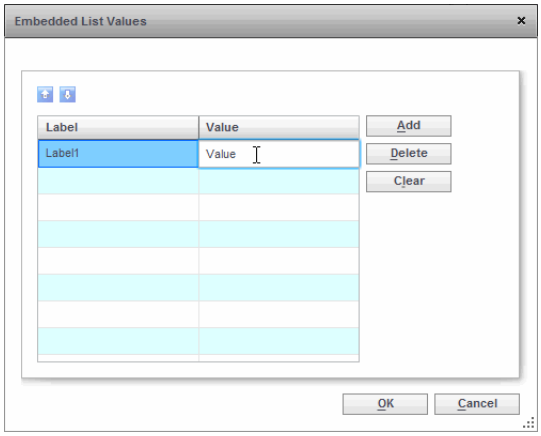
2. Select the **General** tab.

3. Under the heading 'List Values', select the 'Embedded' option, and click on the 'Edit' button. This opens the 'Embedded List Values' dialog box.

4. Click the 'Add' button. This will populate the list with 'Label' and 'Value'.

5. Double click on the list elements, and enter a desired Label and Value.

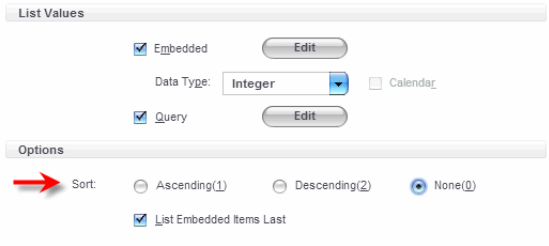
6. To insert more elements, repeat the above steps.



7. Click ‘OK’ when you are finished adding items.

8. In the ‘Data Type’ field, specify the data type of the values you entered in the ‘Value’ column.

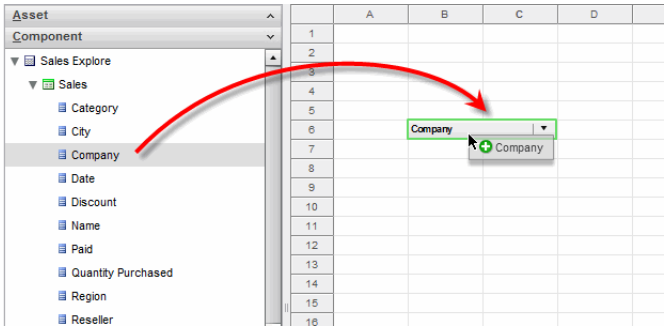
9. In the ‘Options’ panel, specify how the labels in the list should be ordered. ‘Ascending’ and ‘Descending’ sort the labels in alphabetical/increasing order or reverse-alphabetical/decreasing order, respectively.



The ‘None’ option retains the original ordering of the embedded list. If you are populating the list from both a query and an embedded list (see *Populating Label/Value Pairs from Both a Query and Embedded List*), select ‘List Embedded Items Last’ to place the items from the embedded list following the query items. Otherwise, the embedded items are listed before the query items.

10. Press ‘OK’ to exit the ‘Properties’ dialog box.

To use data field names as the label/value pairs for the input control, simply drag the field names from the **Component** panel onto the input component in the desired sequence.



This approach is preferable to manually typing the field names in the embedded list, because it protects you from entering misspelled field names. After adding the fields to the list this way, you can modify the labels by editing the embedded list as described previously.

Populating Label/Value Pairs from Query

Applies to Radio Buttons, Check Boxes, and Combo Boxes.

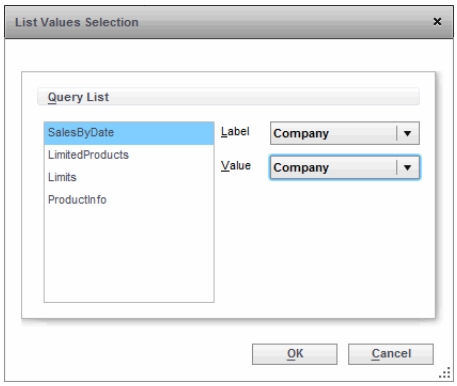
To populate a Radio Button, Check Box, or Combo Box using label/value pairs from a query, follow the steps below:

1. Right-click on the input control, and select ‘Properties’ from the context menu. This opens the ‘Properties’ dialog box.

2. Select the **General** tab.

3. Under the heading ‘List Values’, select the ‘Query’ option, and click on the ‘Edit’ button. This opens the ‘List Values Selection’ dialog box.

4. Select a Data Block from the list on the left side. Select the appropriate fields from the drop-down lists titled ‘Label’ and ‘Value’.

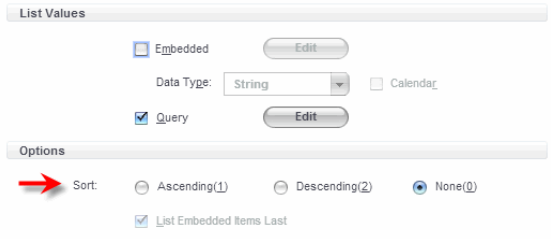


5. Enable the ‘Apply Selection’ option to filter the query values based on Selection components in the Viewsheet. For example, if an input control displays query values drawn from the ‘Category’ field, and

the Viewsheet also contains a Selection List based on the 'Category' field, selections that the user makes in the 'Category' Selection List will filter the categories shown in the input component.

6. Press 'OK' to close the 'List Values Selection' dialog box.

7. In the 'Options' panel, specify how the labels in the list should be ordered. 'Ascending' and 'Descending' sort the labels in alphabetical/increasing order or reverse-alphabetical/decreasing order, respectively. The 'None' option retains the original ordering provided by the query.

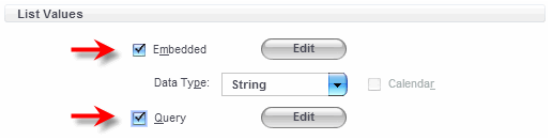


8. Press 'OK' to close the 'Properties' dialog box.

Populating Label/Value Pairs from Both a Query and Embedded List

Applies to Radio Buttons, Check Boxes, and Combo Boxes.

To populate a Radio Button, Check Box, or Combo Box using label/value pairs from both a query and an embedded list, enable both the 'Query' and 'Embedded' options in the 'Properties' dialog box. (See *Populating Label/Value Pairs from Query* and *Populating Label/Value Pairs Manually*.)



This will merge together the labels/values retrieved from both sources into a single list. If the 'Sort' property is set to 'Ascending' or 'Descending', the query items and embedded items are sorted together as a single list. If the 'Sort' property is set to 'None', the embedded items can be placed either before or after the query items. See *Populating Label/Value Pairs Manually* for more information.

Acquiring Date Input Using a Calendar Control

Applies to Combo Boxes only.

To acquire date input from the user using a calendar control, follow the steps below:

1. Drag a ComboBox from the **Components** panel to the Viewsheet.

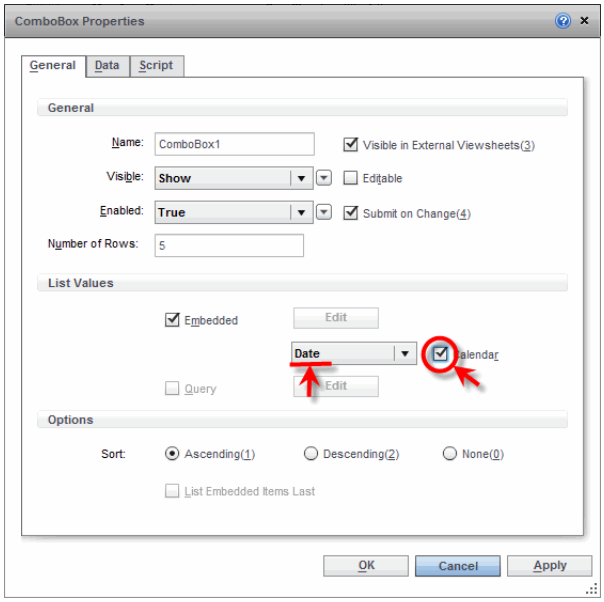
1. Right-click the ComboBox, and select ‘Properties’ from the context menu. This opens the ‘Properties’ dialog box.

2. Select the **General** tab.

3. Under the heading ‘List Values’, select the ‘Embedded’ option, and click on the ‘Edit’ button.

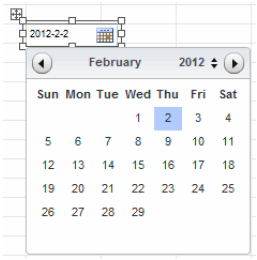
4. In the ‘Data Type’ field, select the ‘Date’ option.

5. Enable the ‘Calendar’ option.



6. Press ‘OK’ to exit the dialog box.

This creates a calendar control in which user the user can select an arbitrary date.



7.3.2 Radio Button/Check Box/Combo Box Properties

The ‘Properties’ dialog box for Radio Button, Check Box, or Combo Box provides **General** and **Data** tabs. The next sections discuss the input-specific properties available under these tabs.

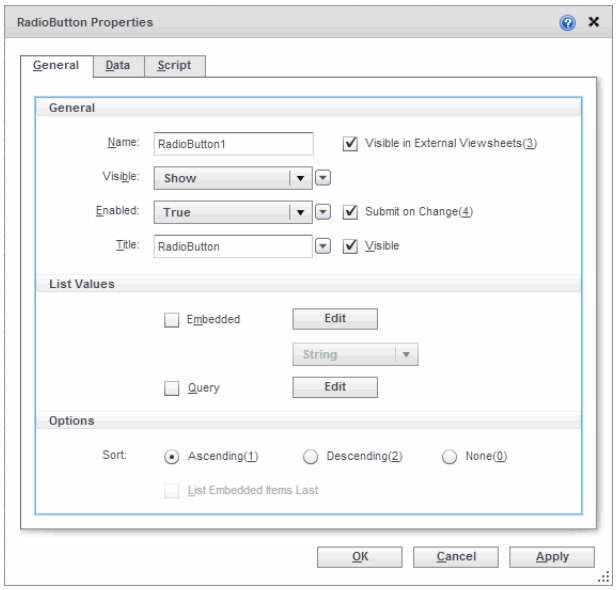
See Also

Properties Dialog Box, for information about common component properties.

Radio/Check/Combo Properties: General Tab

The **General** tab in the 'Properties' dialog box provides the following unique properties for Radio Button, Check Box, and Combo Box.

Submit on Change	When enabled (default), submits the selected value immediately upon selection. Otherwise, a separate 'Submit' button is required.
List Value	Specifies the method for populating control labels and values: **Embedded**: Manually enter label-value pairs. **Query**: Populate from a query (Data Block). See *Creating a Radio Button/Check Box/Combo Box* for information on how to use these options.
Editable	Allows the user to enter an arbitrary string into the Combo Box (in addition to selecting from the list options).
Number of Rows	For a ComboBox, this specifies the number of items displayed when the ComboBox is expanded.



See Also

Populating Label/Value Pairs Manually, to manually enter label-value pairs.
Populating Label/Value Pairs from Query, to populate from a query.
General Tab, for information on common properties in the **General** tab.

Radio/Check/Combo Properties: Data Tab

The **Data** tab in the 'Properties' dialog box for Radio Button, Check Box, or Combo Box is the same as for Sliders and Spinners.

See Also

Slider and Spinner Properties: Data Tab, for information on the **Data** tab.

7.4 Submit Button

The Submit button provides a way for the user to trigger a scripted action in the Viewsheet. This action is performed by a script that you place in the 'onClick' handler of the Submit button.

One common use of the Submit button is to write data from an editable table or *TextInput* component to a database. The submitted table data may include records that the user has added, deleted, or edited in a table.

For information on how to create a database update script, see *Committing User-Modified Data to Database (Database Write-Back)* in *Advanced Dashboard Scripting*.

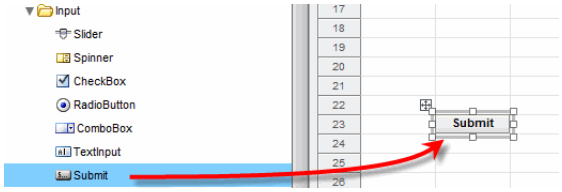
See Also

confirm(message), in *Advanced Dashboard Scripting*, to confirm user submissions using script.
Committing User-Modified Data to Database (Database Write-Back) in *Advanced Dashboard Scripting*, to submit information to the database.
TextInput, for a component that allows arbitrary user text entry and validation.
Table Properties: Advanced Tab, to configure a table to be editable by users.

7.4.1 Creating a Submit Button

To create a Submit button, follow the steps below.

1. Drag the 'Submit' element from the component tree into the Viewsheet grid. This creates a new Submit button in the Viewsheet grid.



2. Right-click on the Submit button, and select 'Properties' from the context menu. This opens the 'Submit Properties' dialog box.

3. Select the **General** tab. In the 'Label' field, enter the text to appear on the button face.

4. Select the **Script** tab. Enter the desired 'onClick' script. See *Committing User-Modified Data to Database (Database Write-Back)* in *Advanced Dashboard Scripting* for more information.

5. Press 'OK' to close the dialog box.

7.4.2 Submit Button Properties

The 'Properties' dialog box for Slider and Spinner components provides the following tabs: **General** and **Script**. The next sections discuss the input-specific properties available under these tabs.

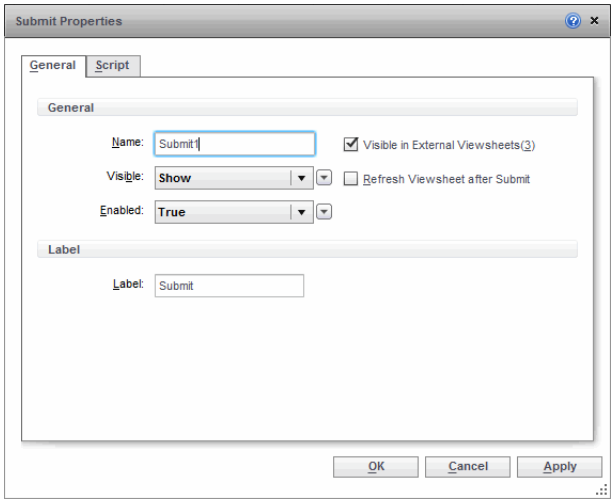
See Also

Properties Dialog Box, for information about common component properties.

Submit Button Properties: General Tab

The **General** tab in the Submit button 'Properties' dialog box provides the following unique property.

Label	The text to be displayed on the button face.
Refresh Viewsheet after Submit	Refresh the Viewsheet data when the user presses the 'Submit' button.



See Also

General Tab, for **General** tab properties that are common among components.

Submit Button Properties: Script Tab

The **Script** tab is common to most components. See *Advanced Dashboard Scripting* for information on writing Viewsheet scripts.

7.5 TextInput

The TextInput component allows a user to enter arbitrary data in the form of text, date, integer, or floating point values. This component is commonly used together with the *Submit Button* to commit user edits to the database or assign values to parameters.

See Also
Submit Button, for information about submitting user input.

7.5.1 TextInput Properties

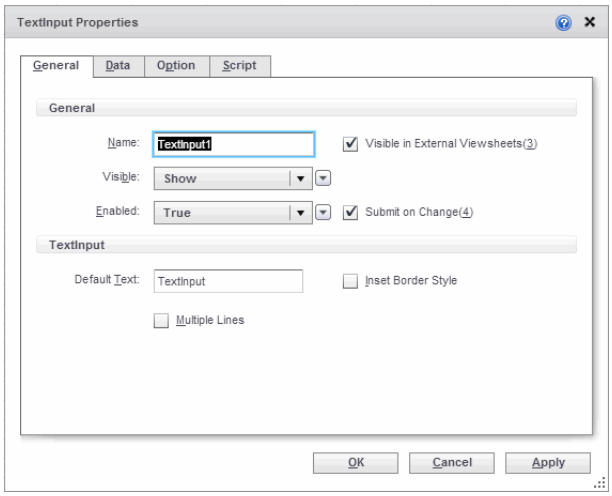
The 'Properties' dialog box for the TextInput component provides the following tabs: **General**, **Option**, and **Script**. The next sections discuss the input-specific properties available under these tabs.

See Also
Properties Dialog Box, for information about common component properties.

TextInput Properties: General Tab

The **General** tab in the 'Properties' dialog box provides the following unique properties.

Submit on Change	When enabled (default), submits the entered value when user presses 'Enter' or clicks away. Otherwise, a separate 'Submit' button is required.
Default Text	Specifies the text to appear within the TextInput field when it is first displayed. (This is often a hint to the user about what data should be entered). This default text is automatically removed when the user begins entering data.
Inset Border Style	Specifies a dimensional "inset" (rather than flat) appearance for the component border.
Multiple Lines	When enabled, allows the user to enter multiple lines of text into the text area.



See Also

General Tab, for **General** tab properties that are common among components.

TextInput Properties: Option Tab

The **Option** tab in the 'Properties' dialog box allows you to select an "Input Editor" which specifies the control interface and validation for user input. See *Validating User Input* for details on the available editors.

TextInput Properties: Script Tab

The **Script** tab is common to most components. See *Advanced Dashboard Scripting* for information on writing Viewsheet scripts.

7.6 Upload

The Upload component allows a user to upload a local file to the server. A script function, *getFileName()*, provides the full path to the uploaded file on the server. A script function, *isLoaded()*, provides verification that a file was uploaded.

7.6.1 Upload Properties

The 'Properties' dialog box for the Upload component provides the following tabs: **General** and **Script**. The next sections discuss the input-specific properties available under these tabs.

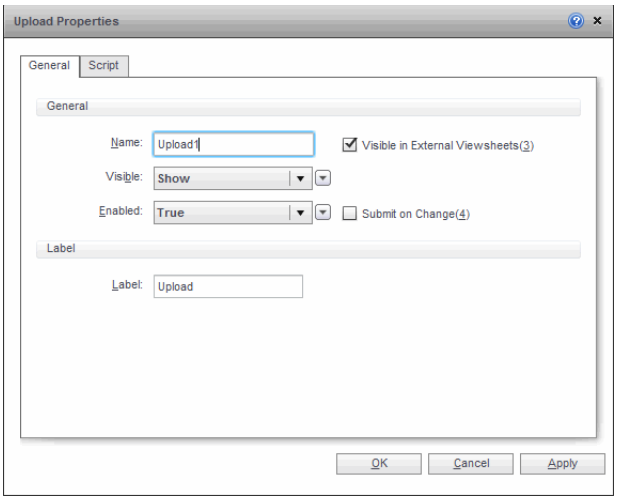
See Also

Properties Dialog Box, for information about common component properties.

Upload Properties: General Tab

The **General** tab in the 'Properties' dialog box provides the following unique properties.

Submit on Change	When enabled (default), uploads the selected file immediately. Otherwise, a separate 'Submit' button is required.
Label	Specifies the label to appear on the button.



See Also
General Tab, for **General** tab properties that are common among components.

Upload Properties: Script Tab

The **Script** tab is common to most components. See *Advanced Dashboard Scripting* for information on writing Viewsheet scripts.

See Also
Upload, in *Advanced Dashboard Scripting*, for related script functions.

8 Shape Components

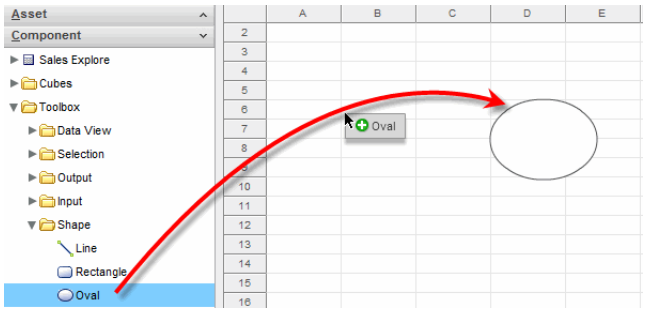
Note: Shapes cannot be exported to Excel format.

Shape components allow you to add simple geometric shapes to a Viewsheet. You can use these shapes to provide frames and backgrounds for other components, and to improve the aesthetic appearance of the Viewsheet.

8.1 Creating a Shape

You can draw several shapes on the Viewsheet grid, including lines, rectangles, and ovals. To draw a shape on the Viewsheet grid, follow the steps below:

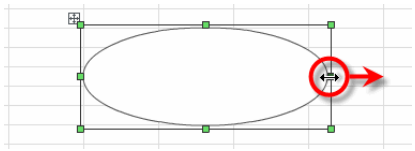
1. From the 'Shape' node on the Component tree, drag the desired shape onto the Viewsheet grid.



2. Click the shape to select it.

3. Drag the body of the shape (or the "move" handle) to reposition the shape as desired.

4. Drag the border handles to resize the shape as desired.



See Also

Creating an Image, for information on adding images to the Viewsheet.

Creating a Background, for information on creating a Viewsheet background.

8.2 Shape Properties

The ‘Properties’ dialog box for shape components provides **General** and **Shape** tabs. The next sections discuss the shape-specific properties available under these tabs.

See Also

Properties Dialog Box, for information about common component properties.

8.2.1 Shape Component Properties: General Tab

The **General** tab in the ‘Properties’ dialog box for shape components provides only the common properties.

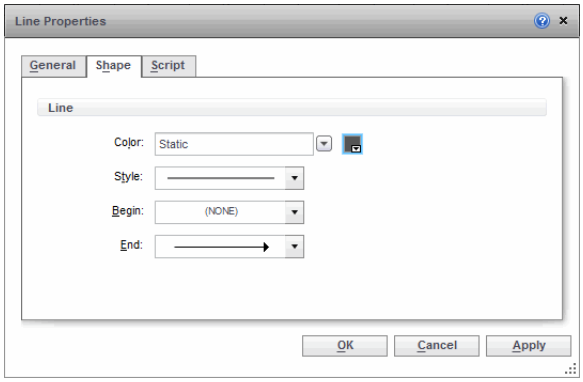
See Also

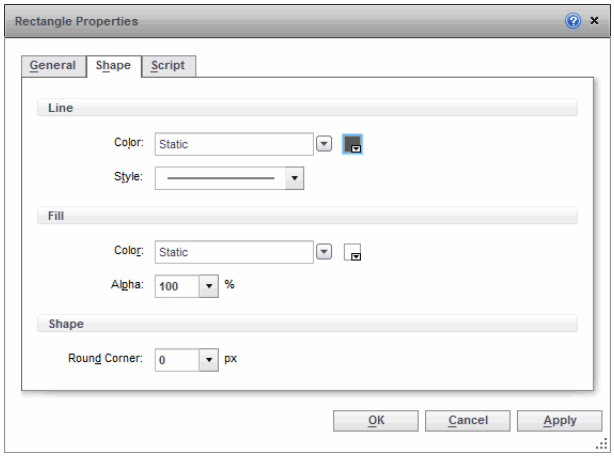
General Tab, for **General** tab properties that are common among components.

8.2.2 Shape Component Properties: Shape Tab

The **Shape** tab in the ‘Properties’ dialog box for shape components provides slightly different properties for each shape type, as explained below

Line: Style Line: Color	Sets the line type (thickness, etc.) and color for shape borders. Available for all shapes.
Fill: Color Fill: Alpha	Sets the color and transparency of Oval and Rectangle shape interiors. An ‘Alpha’ value of 100% indicates complete opacity. An ‘Alpha’ value of less that 100% allows underlying components to be partially visible through the shape.
Shape: Round Corners	Sets the radius (in pixels) for circularized corners on Rectangle shapes. Larger pixel values yield more gentle corners.





8.3 Anchoring a Line to another Component

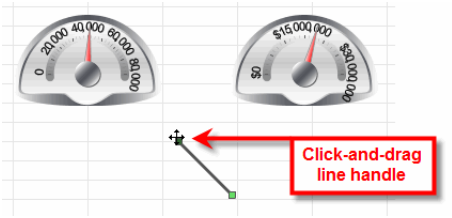
You can *anchor* or *glue* the end of a Line component to a handle of any other component. This is useful when you are using the line as a *connector* between components. When you anchor the end of a line, it remains attached to the anchoring component even when the anchoring component is moved.

To anchor the end of a Line component to another component, follow the steps below:

1. Drag a new Line component from the **Components** panel into the Viewsheet.

2. Set the Line properties as desired. (See *Shape Component Properties: Shape Tab*.)

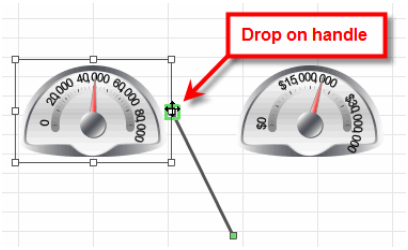
3. Click on one of the handles of the Line that you want to anchor.



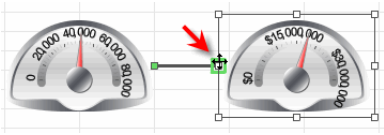
4. Drag the Line handle over the component to which it should be anchored. This causes the handles of the anchoring component to be displayed.

Note: A green highlight is shown around the handle when the end of the Line is in the correct position.

5. Drop the Line handle onto one of the handles of the anchoring component.



6. (Optional) Repeat the above steps to anchor the other end of the Line.



The end of the Line that is anchored will remain attached to the anchoring component even when the anchoring component is moved.

To free the end of the line from its anchor, follow the steps below:

1. Click the Line to select it.
2. Click the handle at the end of the Line, and drag it away from its anchor.

9 Organization Components

There are several components whose function is to organize other components, allowing you to structure the components within a Viewsheet. These organizational components are listed below:

- *Container*: The Container component is used to group a set of components, allowing you to positioned them together as a single unit.
- *Tabbed Interface*: The Tabbed Interface component is used to stack components onto distinct pages (accessed by tabs), allowing you to make more efficient use of Viewsheet real estate.
- *Nesting Viewsheets*: The Nested Viewsheet component is created when you import an external Viewsheet. This allows you to modularize your dashboard design.

The following sections present detailed information about these components' properties.

See Also
Grouping Components, for information on how to create a group.
Selecting Components, for information on how to select multiple components.

9.1 Container

The Container component is a container for other elements. It is automatically created whenever you *group* a set of components together. See *Grouping Components*.

The Container component can hold only individual components, not other Containers. If you group together multiple Containers, the result is a single new Container that holds all of the constituent group's components.

9.1.1 Creating a Container

The Container component is not accessible from the Viewsheet Component tree. To create a Container component, simply *group* together a set of Viewsheet components. See *Grouping Components* for further details.

9.1.2 Container Properties

The Container component's 'Properties' dialog box provides **General** and **Advanced** tabs. The next sections discuss the Container-specific properties available under these tabs.

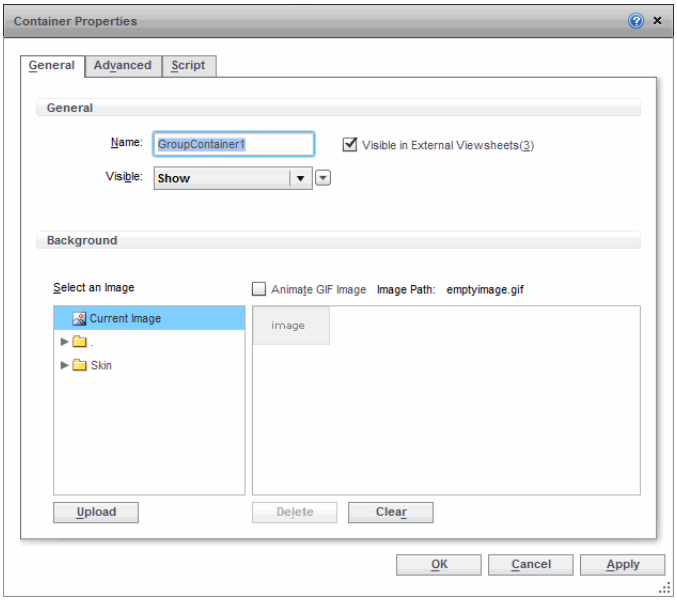
See Also

Properties Dialog Box, for information about common component properties.

Format Dialog Box, for information on the 'Format' dialog box.

Container Properties: General Tab

The **General** tab in the 'Container Properties' dialog box is the same as that for the *Image* component, and allows you also to specify an image to be used as the group background. See *Image Properties: General Tab* for details on how to select the image.



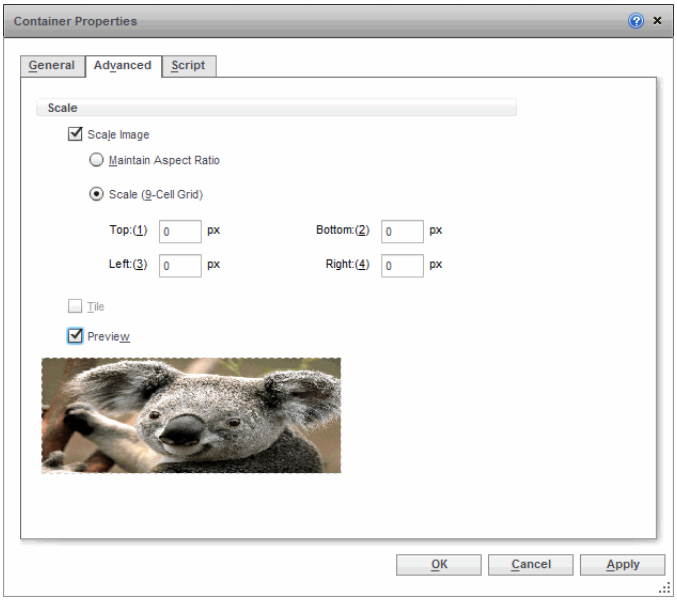
See Also

Format Dialog Box, for information on setting background colors.

General Tab, for information about common properties in the **General** tab.

Container Properties: Advanced Tab

The **Advanced** tab in the 'Container Properties' dialog box is the same as that for the *Image* component, and allows you to set the scaling properties of the group background image. See *Image Properties: Advanced Tab* for details on image scaling.



9.2 Tabbed Interface

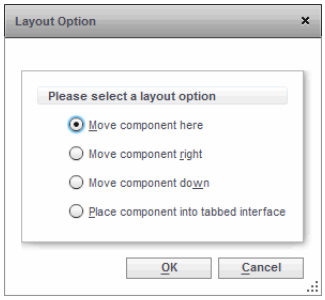
A Tabbed Interface allows you to stack individual components or groups into distinct cards or pages, accessed by tabs. This is an effective way to conserve screen real estate if components do not need to be accessible at the same time.



9.2.1 Creating a Tabbed Interface

The Tabbed Interface component is not accessible from the Viewsheet Component tree. To create a tabbed interface, follow the steps below.

1. Drag and drop one component (or group) onto another component (or group). This opens the ‘Layout Option’ dialog box.



2. Choose the 'Place component into tabbed interface' option. This places the two components (or groups) into a tabbed interface.



Note that each component is placed at the top-left corner within its own tab. To display a component at a different location within its tab, first group the component together with a background (e.g., an *Image* component or one of the *Shape Components*), and then use the group as the contents of the tab. See *Grouping Components* for more information.

Any of the following components can be placed into a tabbed interface:

- Individual components (e.g., Calendar, Table, Chart, Selection Container)
- Grouped components (i.e., a Container element)
- Nested Viewsheets

To add an additional component to an existing tabbed interface, simply drag the component (or group) onto the tabbed interface. This will create a new tab containing the component (or group).

To remove a particular tab from the tabbed interface, follow these steps:

1. Click the tab you wish to remove.
2. Select the component or group within that tab.
3. Drag the selected component out of the tabbed interface (or Cut and Paste) onto an empty region of the Viewsheet. This will

You can also delete the component, if desired.

remove the component from the interface (and delete the corresponding tab).

To remove the entire tabbed interface, follow the steps below:

1. Click on the row of tabs to select the tabbed interface (as a whole).

2. Right-click, and select ‘Remove’ from the context menu (or click ‘Delete’ on the keyboard).

This deletes the tabbed interface, and returns all of the tabbed components to normal positions on the Viewsheet.

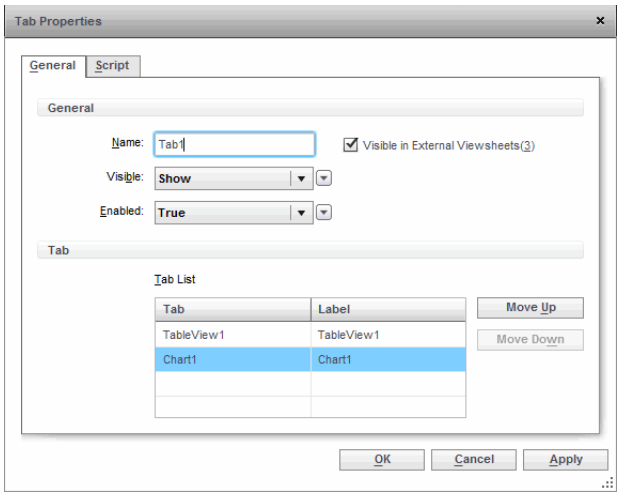
See Also

Grouping Components, for information on how to create a group.
Selecting Components, for more information on how to select a component.
Deleting a Component, for information on how to remove a component.
Moving a Component, for information on moving an element.

9.2.2 Tabbed Interface Properties

Note: To open the Tabbed Interface ‘Properties’ dialog box, right-click in the tab region, and select ‘Properties’.

The Tabbed Interface component’s ‘Properties’ dialog box provides only a **General** tab. This allows you to assign labels to the different tabs and modify their order.



To change a tab label, select the desired tab in the ‘Tab’ list, enter the new label text in the ‘Label’ field, and click ‘OK’. To change the position of the selected tab, click ‘Move Up’ or ‘Move Down’.

See Also

Properties Dialog Box, for information about common component properties.
Format Dialog Box, for information on setting background colors.

General Tab, for information about common properties in the **General** tab.

9.3 Nesting Viewsheets

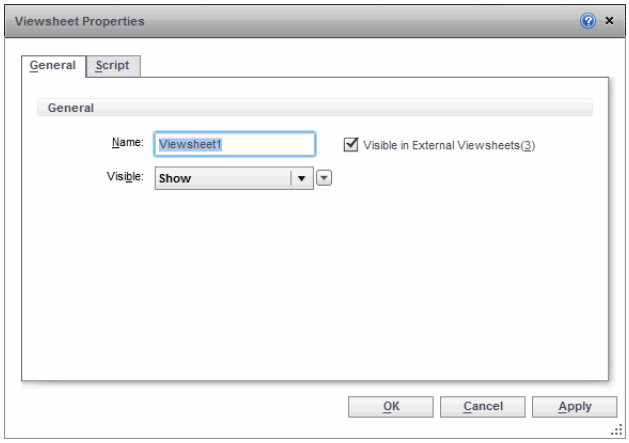
A nested Viewsheet is an external Viewsheet that is embedded within the current Viewsheet. This nested Viewsheet can be positioned within the current Viewsheet as though it were a single component, and it has its own 'Properties' dialog box.

See *Nesting a Viewsheet* for information on creating a Nested Viewsheet component.

See Also
Nesting a Viewsheet, for information on embedding Viewsheets.
Opening a Nested Viewsheet, for information on editing a nested Viewsheet.

9.3.1 Nested Viewsheet Properties

The nested Viewsheet's 'Properties' dialog box provides only a **General** tab. See *General Tab* for information on these common properties.



See Also
Properties Dialog Box, for information about common component properties.
General Tab, for information about common properties in the **General** tab.

10 Dynamic Properties

Every component in a Viewsheet has certain properties that can be made "dynamic," or dependent on other components. These properties can be set either by an input component or by a logical expression. For example, you can use a radio button to specify the Y-axis field in a chart, or use an expression to add highlights to cells of a table.

The following sections explain how you can use input controls to parameterize component formatting and data presentation, as well as data filtering. By associating an input control such as a RadioButton or ComboBox with the property setting for a component, you allow the user to adjust that setting by manipulating the corresponding control.

For information on using expressions, see *Adding Property Script (Expressions)* in *Advanced Dashboard Scripting*.

See Also
Adding Property Script (Expressions), in *Advanced Dashboard Scripting*, to control property values with expressions.

10.1 Parameterizing a Property using an Input Control

To parameterize a component property using an input control, follow these steps:

1. Create and configure the input control. Follow the steps below:

    a. Add the desired input control (RadioButton, CheckBox, etc.) to the Viewsheet.

    b. Right-click the input component, and select 'Properties' from the context menu. This opens the 'Properties' dialog box for the input component.

    c. Under the **General** tab, use the 'Name' field to assign the input component a meaningful name. This is the name used to refer to the component.

    d. Specify the 'List Values' for the input component by manually entering 'Embedded' values or by specifying a 'Query' to provide the values. (See *Input Components* for further details.)

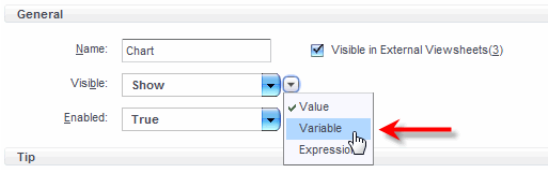
    e. Click 'OK' to close the input component 'Properties' dialog box.

2. Link the input control to the component property you want to manipulate. Follow the steps below:

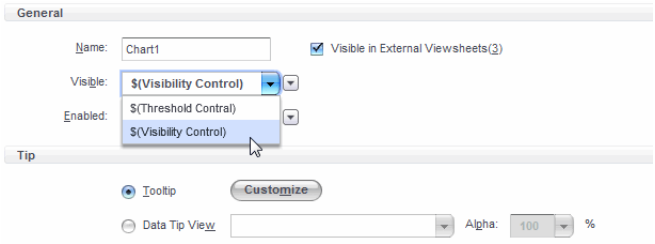
   a. Right-click the component you wish to control, and select 'Properties' from the context menu. This opens the 'Properties' dialog box for the controlled component.

You can specify an input control for any property that offers the 'Variable' option.

   b. Locate the property that you wish to control, and select 'Variable' from the adjacent pop-up menu.



   c. From the corresponding pop-up menu, select the input component that should control the property.



   d. Click 'OK' to close the 'Properties' dialog box for the controlled component.

The specified input control will now provide the value of the selected property.

Example: Property as a Parameter

In this example, you will add a RadioButton input that allows the user to set the Y-axis field of a chart to either 'Quantity Purchased' or 'Total Revenue'.

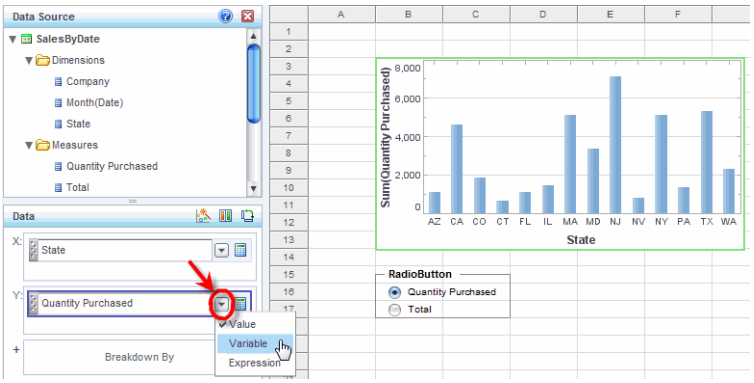
1. Create a Viewsheet based on the 'ProductSales' Data Worksheet in the 'Tutorial' folder.

2. Add a radio button to the Viewsheet, with two embedded values ('Quantity Purchased' and 'Total').

3. Add a chart to the Viewsheet and bind it to the 'SalesByDate' Data Block. Add the 'State' field as the X-axis and the 'Quantity Purchased' field as the Y-axis.

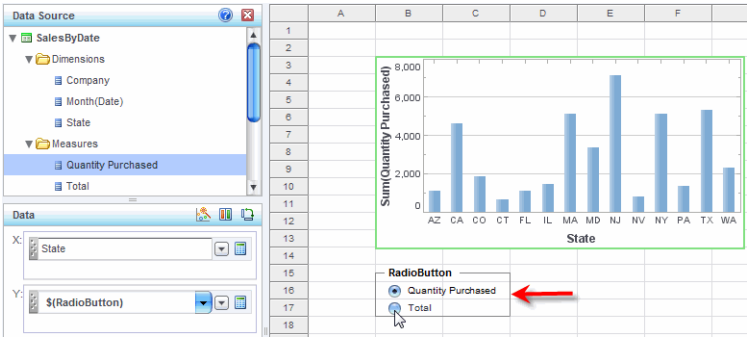
4. Click the ‘Quantity Purchased’ menu to select it.

5. Click the ‘down arrow’ next to the menu, and choose the ‘Variable’ option. This populates the menu with all of the available variable names.



6. From the list of variables, choose ‘$(RadioButton)’.

7. Now click the radio buttons. Note that the chart displays the Y-axis field that you choose using the radio button control.



¢

See Also

Matching Dynamic Values to Property Values, for input value constraints.

10.2 Parameterizing a Condition using an Input Control

Input components allow a user to manipulate a filter or highlight condition applied to a data view component (Chart, Table, Crosstab). To control such a condition using an input component, follow these steps:

1. Create and configure the input control, following the steps below:

a. Add the desired input control (RadioButton, CheckBox, etc.) to the Viewsheet.

b. Right-click the input component, and select ‘Properties’ from the context menu. This opens the ‘Properties’ dialog box for the input component.

c. Under the **General** tab, use the ‘Name’ field to assign the input component a meaningful name. This will be the name used to refer to the component.

d. Specify the ‘List Values’ for the input component by manually entering ‘Embedded’ values or by specifying a ‘Query’ to provide the values. (See *Input Components* for further details.)

e. Click ‘OK’ to close the input component ‘Properties’ dialog box.

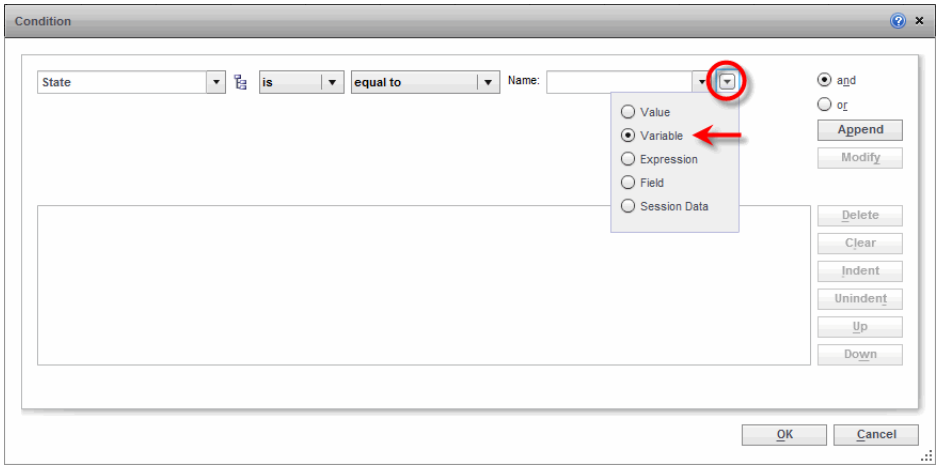
2. Link the input control to the filter condition or highlight condition of the data view component that you want the user to manipulate. Follow the steps below:

a. Right-click the data view component you wish to highlight or filter, and select ‘Highlight’ or ‘Condition’ (as desired) from the context menu. This opens the ‘Highlight’ or ‘Condition’ dialog box for the component.

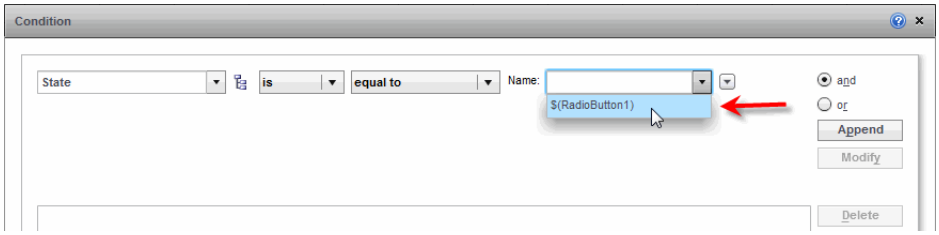
b. Create the left side of the condition as described under *Highlights* and *Filter Conditions*, e.g., “`[State][is][one of]`”.



c. From the ‘Change Value Type’ popup menu (triangle), select ‘Variable’. This enables an adjacent pop-up menu.



d. From the adjacent pop-up menu, select the input component that you want to control the value.



e. Click 'Append' to add the condition, and click 'OK' to close the 'Condition' dialog box.

The specified input control will now provide the value for the right side of the condition.

Example: Condition as a Parameter

In this example, you will add a CheckBox input that allows the user to highlight specified states within a table.

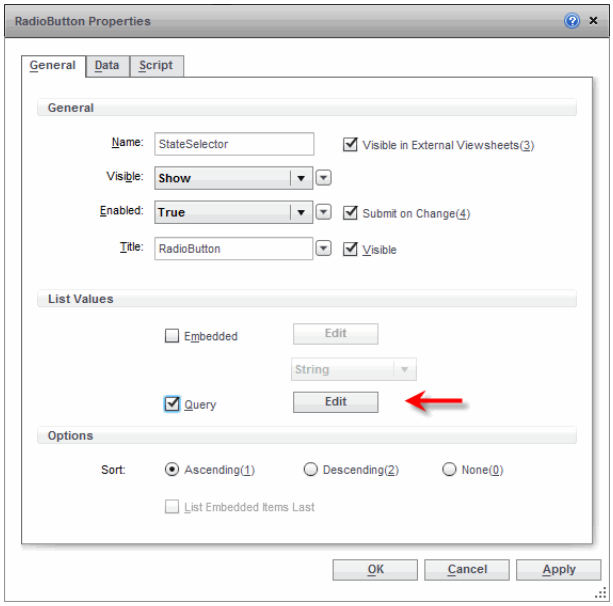
1. Create a new Viewsheet based on the 'Sales Explore' Worksheet (in the 'Sales' folder).

2. Drag a CheckBox component from the **Component** panel to the Viewsheet. Right-click the component, and select 'Properties' from the context menu. This opens the 'CheckBox Properties' dialog box.

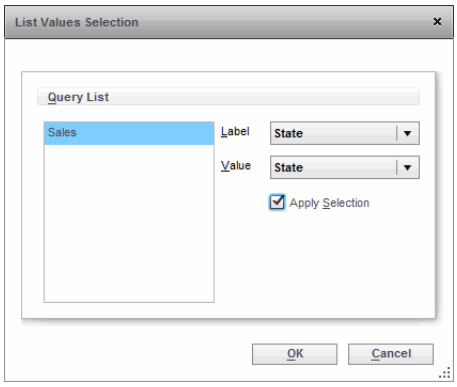
3. In the 'CheckBox Properties' dialog box, make the following changes:

   a. Change the 'Name' to "StateSelector".

   b. In the 'List Values' panel, select the 'Query' option. Click the 'Edit' button to open the 'List Values Selection' dialog box.

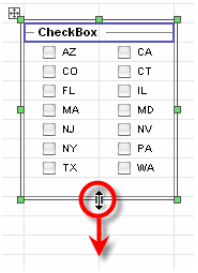


c. In the 'List Values Selection' dialog box, select the 'Sales' Data Block in the left panel, and then select 'State' in both the 'Label' and 'Value' menus.



d. Click 'OK' to close both dialog boxes. This binds the CheckBox component to the 'State' field of the Data Block.

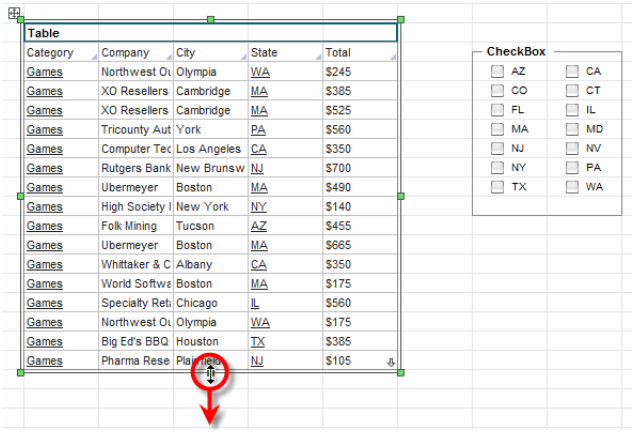
4. On the Viewsheet grid, select the CheckBox component. Drag the handles to expand the component so that all state options are visible.



5. Drag a Table component from the **Component** panel to the Viewsheet grid.

6. Expand the ‘Sales’ Data Block at the top of the **Component** panel, and drag the following fields into the Table: ‘Category’, ‘Company’, ‘City’, ‘State’, ‘Total’.

7. Enlarge the table so that at least 10 rows are visible.

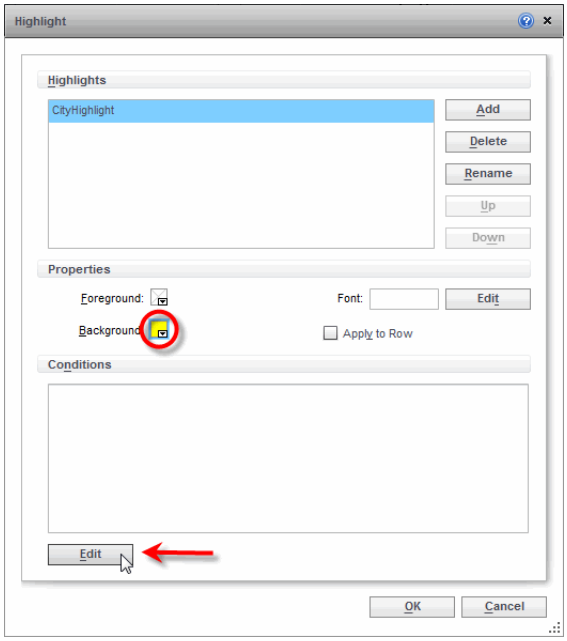


8. Right-click a cell in the ‘City’ field of the Table, and select ‘Highlight’ from the context menu. This opens the ‘Highlight’ dialog box for the ‘City’ field.

9. In the ‘Highlight’ dialog box, click the ‘Add’ button, and enter the name “CityHighlight”.

10. For the ‘Background’ property, select yellow.

11. In the ‘Conditions’ panel, click the ‘Edit’ button to open the ‘Conditions’ dialog box.

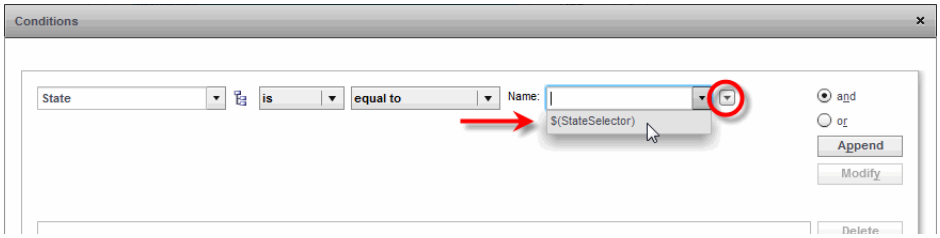


12. In the ‘Conditions’ dialog box, create the left side of the condition by making the following menu selections: “`[State][is][one of]`”.



13. From the ‘Change Value Type’ popup menu (triangle), select ‘Variable’. This enables an adjacent pop-up menu.

14. From the adjacent pop-up menu, select the “StateSelector” input component.



15. Click ‘Append’ to add the condition, and click ‘OK’ to close the ‘Conditions’ dialog box.

16. Click ‘OK’ to close the ‘Highlight’ dialog box.

On the Viewsheet, make selections in the CheckBox input component, and observe how this highlights the cities in the Table that correspond to the selected states.

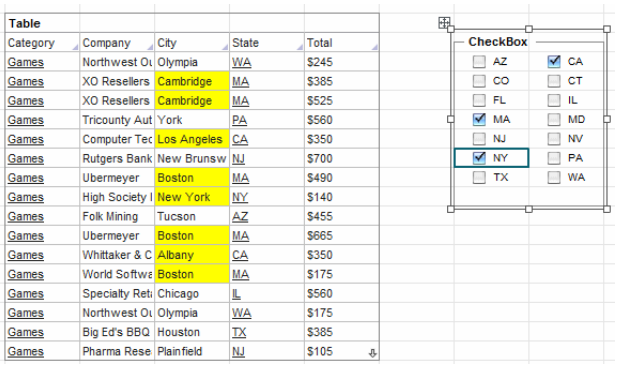


Table				
Category	Company	City	State	Total
Games	Northwest Ou	Olympia	WA	$245
Games	XO Resellers	Cambridge	MA	$385
Games	XO Resellers	Cambridge	MA	$525
Games	Tricounty Aut	York	PA	$560
Games	Computer Tec	Los Angeles	CA	$350
Games	Rutgers Bank	New Brunsw	NJ	$700
Games	Ubermeyer	Boston	MA	$490
Games	High Society I	New York	NY	$140
Games	Folk Mining	Tucson	AZ	$455
Games	Ubermeyer	Boston	MA	$665
Games	Whittaker & C	Albany	CA	$350
Games	World Softwa	Boston	MA	$175
Games	Specialty Reti	Chicago	IL	$560
Games	Northwest Ou	Olympia	WA	$175
Games	Big Ed's BBQ	Houston	TX	$385
Games	Pharma Rese	Plainfield	NJ	$105

¢

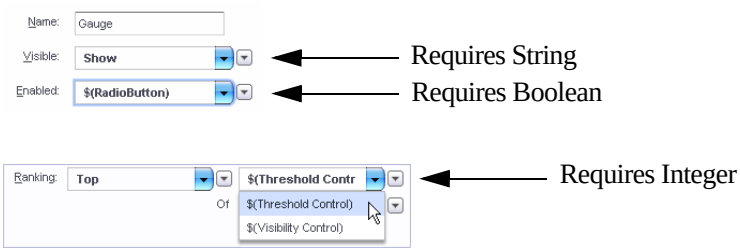
See Also

Matching Dynamic Values to Property Values, for input value constraints.

10.3 Matching Dynamic Values to Property Values

When you link an input control with a component property, you should ensure that the input control evaluates to a permissible value for the particular property. There are three main cases.

- **Property Requires a Numerical Value**: If a property requires a numerical value, the input control should provide an appropriate numerical value. For example, the ‘Ranking’ property in the Chart data binding requires an integer value. An appropriate control for this property would therefore be a Spinner with ‘Increment’ property set to an integer value.
- **Property Requires a Boolean Value**: If a property has a ‘True’ or ‘False’ setting, the input control should provide a Boolean value. For example, the ‘Enabled’ property expects a Boolean value ‘true’ or ‘false’. An appropriate control for this property would therefore be a RadioButton with embedded list values {true, false}, and with ‘Data Type’ set to ‘Boolean’.
- **Property Requires a String Value**: If a property requires one of a fixed set of strings, the input control should provide a string from the set. For example, the ‘Visible’ property requires one of the strings ‘Show’, ‘Hide’, or ‘Hide on Print and Export’. The input control must therefore return one of these strings. An appropriate control for this property would therefore be a RadioButton with embedded list values {‘Show’, ‘Hide’, ‘Hide on Print and Export’}, and with ‘Data Type’ set to ‘String’.


Name:
Gauge
Visible:
Show
Enabled:
$(RadioButton)
Requires String
Requires Boolean
Ranking:
Top
$(Threshold Contr
Of
$(Threshold Control)
$(Visibility Control)
Requires Integer

Advanced Dashboard Scripting

Certain dashboard applications require you to implement a degree of intelligence and adaptability that would be difficult or impossible to design using only basic Viewsheet components. For example, you may wish to hide or show certain dashboard components under particular conditions, such as when a user's selection return no data. Likewise, you might wish to present an alert message if a certain key indicator does not meet its target.

For these purposes, Style Intelligence provides an embedded scripting environment for Viewsheets. The scripting environment allows you to implement event-driven logic that adaptively modifies dashboard appearance and behavior based on retrieved data, inputs from users, or other factors such as day, time, locale, etc. The scripting language is intended to be simple but powerful, requiring only minimal programming knowledge.

The scripting language is based on the ECMA-262 (JavaScript) standard. Since there are many outstanding books on JavaScript and the full specification of the language is available online, this Guide does not attempt to be a JavaScript reference. Instead, it provides a short introduction to JavaScript, but concentrates on the host Viewsheet environment and examples of common Viewsheet scripting.

1 Introduction to JavaScript Programming

Netscape originally introduced JavaScript as the scripting language for its Navigator web browser. The scripting language allows code to be embedded in HTML pages and executed inside a browser, which acts as the host environment for the scripts. The browser exposes HTML page elements and browser controls to the scripts as host objects and functions. By manipulating these host objects, JavaScript is able to control the behavior of the browser and add interactivity to the web pages.

Since the release of Netscape Navigator 2.0 and Microsoft Internet Explorer 3.0, JavaScript has established itself as the standard browser scripting language. Subsequent standardization efforts have produced the specification known as ECMA-262, which is the standard implemented in Style Intelligence.

As a standardized scripting language, JavaScript provides an excellent foundation for report scripting needs. Its origin as a web scripting language makes it ideal for the reporting environment. The integration of JavaScript with Style Intelligence therefore combines two powerful paradigms for report generation and gives developers maximum flexibility in controlling reports and user interactions.

The purpose of this section is to provide a broad overview of JavaScript programming, and how JavaScript can be embedded in Viewsheets. A simple JavaScript reference is provided in Appendix JS: *General JavaScript Functions*. To find more complete coverage, please refer to a JavaScript book or tutorial website (e.g., http://www.w3schools.com/js).

If you are already familiar with the JavaScript language, you can skip ahead to *Adding Script to a Viewsheet*, which explains the different areas of a Viewsheet to which you can add script.

1.1 Server-Side vs. Client-Side Scripting

In Web development, JavaScript is used to add client-side scripting to HTML pages. Scripts are embedded into the contents of a Web page and executed inside the client browser. Traditionally, these client-side scripts deal exclusively with user-browser interactions, and have limited server-side actions (e.g., HTTP requests).

In contrast, Viewsheet scripting is server-side scripting. The scripts contained in a report are executed as part of the report generation process on the server. Style Intelligence scripts can also control certain client-side interactions through event handlers and hyperlinks.

1.2 Object-Oriented Concepts

JavaScript is an object-oriented programming (OOP) language, and provides various objects and methods, as well as the ability to create user-defined methods. To use JavaScript effectively, it is important to understand the following concepts:

- *Properties*: Properties are predefined attributes associated with an object. You can ‘get’ or ‘set’ these properties to observe or alter the corresponding aspects of the object.
- *Methods*: Methods are predefined functions associated with an object. In general, these functions operate on the object itself.
- *Events*: Events are predefined actions that are recognized by an object, such as a mouse movement or clicking.

1.3 JavaScript Language Basics

JavaScript syntax is very similar to that of C++ and Java. It uses the same construct for all loops and has similar syntax for operators. The following is a typical script:

```
var total = 0;

for(var i = 1; i < Table1.table.length; i++) {
   total += parseInt(Table1.table[i][1]);
}

text = (total / Table1.table.length).toFixed(2);
```

This section covers the basics of programming with JavaScript.

Comments and Names

JavaScript uses double slashes ‘//’ to start a single line comment and ‘/* */’ to enclose a multi-line comment.

```
// single line comment
/* multi-line
   comment */
```

The semicolon is used as a statement separator:

```
var n = 0;
k = parseInt(123);
```

Variable names can only contain alphanumeric characters plus the underscore character (_). They cannot start with a digit and cannot use reserved JavaScript words.

All symbols in JavaScript are case-sensitive.

Declaration and Assignment

JavaScript is a weakly-typed language. This means variables are not assigned a type. A variable can be assigned any value. “The type of the variable is determined by the value currently assigned to it. Consequently, a local variable does not need to be declared before it is used.

```
var variable_name= "Hello"; // -- recommended
message1= "Hello"; // -- will also work
var count = 100;
```

If a variable is used as a report level variable, it must be declared using the ‘var’ keyword.

```
// put this in onRefresh handler
var pageTotal = 0;
```

After the variable is declared, it can be used everywhere as a shared report instance variable.

Object Types and Scopes

JavaScript is object-based. This means that every value in JavaScript is an object. As with any Object Oriented language, the properties and methods are associated with each object and need to be invoked by qualifying the names with the object name.

```
// the following statements are equivalent
var name = first_name.concat(last_name);
name = first_name + last_name;
```

As is the case in C++ and Java, if a script is running inside an object scope, it can reference its properties and methods without qualifying the name. There is a global scope in JavaScript, which provides the common methods. Since every script is running inside a global scope, those methods do not need to be qualified.

```
// parseInt() is a global method
var num = parseInt(parameter['count']);
// toFixed() is a number method, so it needs to be
qualified
var int_num = num.toFixed(0);
```

Number Type

JavaScript does not have an integer type and a float type. All numbers are treated as float by default.

```
var total = 2 + 3;
text = total; // this will convert the number to 5.0
```

To force a number to be treated as an integer, use the toFixed() method of number object and give a decimal point of zero:

```
text = total.toFixed(0); // this generates a string 5
```

Number constants can be in decimal format, hexadecimal format if it starts with ‘0x’, and octal format if it starts with ‘0’.

```
decimal_number = 255 // -- Decimal is the default
hex_number = 0xff
fp_number = 2.456 // -- Floating point number
```

Numbers can be used in computation using the usual operators, +, *, /, -. The increment and decrement operators (++, --) are also available in Java.

Boolean Type

A Boolean has a value of true or false.

```
while(true) {
}
```

All undefined values are treated as false Boolean values when used in the context of a condition. You can check if a value is defined by using the value in the if() condition as in

```
if(parameter['start_time']) {
  // do something
}
```

String Type

Enclose String constants with a single or double quote:

```
var mystring = "InetSoft Technology Corp."
mystring = 'InetSoft Technology Corp.'
```

Strings can be concatenated using the plus operator:

```
var str = 'Total: ' + total;
```

If a value concatenated to a string is of a different type, it is converted to a string automatically. Strings have many methods, with the most frequently used including: substring(), toLowerCase(), toUpperCase() and indexOf().

```
var str = 'abcd';
str = str.toUpperCase(); // converts to ABCD
var bIdx = str.indexOf('B'); //return 1
str = str.substring(1, 2); // return 'B'
```

Strings have built-in support for regular expressions. You can find a regular expression in a string using the match() or search() method:

```
str = 'this is a test of regular expression';
re = /test/; // create a regular expression
var idx = str.search(re); // return the position of the
regular expression
```

Date Type

Date is represented as milliseconds since EPOC (Equipment Point of Connection). Creating a date object with no parameter gives the current time:

```
var now = new Date();
```

A date can be converted to a string using a global function, formatDate().

```
str = formatDate(now, 'yyyy-MM-dd'); // 2002-02-21
```

The date format uses the same syntax as in java.text.SimpleDateFormat.

Arrays

An array literally is a list contained in a single variable. Each item in the list is known as an array element and is enclosed in square brackets (`[]`). When you create an array, it is initialized with the specified values as its elements. Its length is set to the number of elements specified. The following example creates the `coffeelist` array with three elements and a length of three:

```
coffeelist = ["French Roast", "Columbian", "Kona"];
```

Multidimensional Arrays are represented by an array of arrays. A multidimensional array (rows and columns) may be created as follows:

```
monthly_rain = [['Jan', 'Feb', 'Mar'],
                [100,10,30],
                [30,10,300],
                [10,10,10]];
```

Conditionals

The if/else statement requires a boolean expression to decide between two alternative actions.

```
if (x > 0) {
  reg = event.region;
}
else {
  reg = event.firstRegion;
}
```

For Loop

A for loop instructs the computer to repeat an action a specific number of times.

```
//for (initial, condition-check, increment)
for (var j = 1; j < 10; j++) {
  total = parseFloat(report['Table'].table[reg.y + j][3]);
}
```

The first expression in the loop initializes the loop variable. The second expression is a condition used to check when to terminate the loop (when it is false). The third expression is the increment that is evaluated at the end of every iteration.

While Loop

The while loop iterates until the condition becomes false.

```
var n = 5;
Text1.text = "The factorial of " + n;
var fact = 1;

// -- Compute n!, where n is a non-negative integer
while (n > 1) {
  fact = fact * n;
  n = n -1;
}

Text1.text += " is " + fact;
```

The break command can be used inside any loop to terminate the loop and ‘continue’ can be used to skip to the next iteration.

Switch Statement

A switch statement chooses a branch to be executed based on a value. One or more values can be listed on each case. A ‘break’ must be included at the end of each case to terminate the switch statement.

```
switch(action) {
case 'A':
  text = 'Add';
  break;
case 'D':
  text = 'Default';
  break;
default:
  text = "N/A";
  break;
}
```

The default label at the end serves as a catch-all phrase. If the switch value does not match any of the listed values, the default section is processed.

Functions

JavaScript, like most programming languages, has facilities for creating subprograms to modularize or divide programs into distinct functions. When required, a function can be called to carry out a particular task.

JavaScript functions behave a little differently than you might expect:

- There is no value reference distinction
- There is no data type distinction

- Return values do not have data types and are optional

```
function max(a,b) // -- Multiple Returns
{
  if (a > b)
    return(a); // -- Return a because it is larger
  else
    return(b); // -- Return b because it is larger
}
```

To create and save a custom JavaScript function, see *Using the Script Library* in *Report Scripting*.

1.4 Useful Text/String Functions

Two common string operations are changing case and searching for substrings.

1.4.1 Changing a String to Upper/Lower Case

To change a string to upper/lower case, use the 'toUpperCase()' and 'toLowerCase()' functions respectively. For example:

```
var s = 'John Lennon';
Text1.text = s.toLowerCase();
```

To change the header cells of a table to upper case, add the following lines to the table-level script:

```
for(var col = 0; col <Table1.table.size; col++) {
  Table1.table[0][col] =
      Table1.table[0][col].toUpperCase();
}
```

1.4.2 Searching Within a String

To find one string within another string, use the 'indexOf()' function. The 'indexOf' function returns the starting index of the substring within the parent string. If the substring is not found, it returns a value of -1. For example:

```
var state = 'New Jersey';

if(state.indexOf('New') > -1) {
  Text1.text = 'With New';
}
else {
  Text1.text = 'Without New';
}
```

1.5 Useful Date Functions

This section discusses several basic date functions.

1.5.1 Finding Difference Between Dates

The 'dateDiff()' function is used to find the difference between two dates in days/months/years.

```
dateDiff('d', fromDate, toDate); // days
dateDiff('m', fromDate, toDate); // months
dateDiff('yyyy', fromDate, toDate); // years
```

For example, if you have a table bound to a query which contains the date field 'Birth Date', you can create a formula field (in the data binding dialog) that calculates the age of this Birth Date in years by subtracting the birthday from the current date. The script of this formula field is shown below.

```
dateDiff('yyyy', field['Birth Date'], CALC.today())
```

1.5.2 Calculating a Past or Future Date

The 'dateAdd()' function is used to find a date which is *n* number of days/ months/years before/after another date.

```
// 1 day before today
dateAdd('d', -1, CALC.today());

// 5 months after today
dateAdd('m', 5, CALC.today());

// 3 years before Order Date
dateAdd('yyyy', -3, field['Order Date']);
```

For example, wish to run a query in script; this query takes in two date parameters namely 'StartDate' and 'EndDate'. You want the query to always fetch data which is 15 days before the today and 15 days after today.

```
var sd =   dateAdd('d', -15, CALC.today());
var ed =   dateAdd('d',  15, CALC.today());
var q = runQuery('Order Bookings', [['StartDate',
sd],['EndDate', ed]]);
```

1.5.3 Extracting a Date Component

When extracting calendar elements within a date (year, month, quarter, etc.) use the date functions within the CALC function library. For example:

```
//extracting the current date and time
var todDate = CALC.today(); //e.g., Feb-21-2007

//extracting the year
var y = CALC.year(todDate); // 2007

//extracting the quarter
var q = CALC.quarter(todDate); // 1

//extracting the day of the week
var dow = CALC.weekdayname(todDate); // Wednesday
```

```
//extracting the date
var d = CALC.day(todDate); // 21

//extracting the month
var m = CALC.month(todDate); // 2
```

1.5.4 Formatting a Date

Date fields can be formatted in script using the 'formatDate()' function. For example:

```
var d = CALC.today();  // e.g., Feb-21-2007
Text1.text = 'Today Is: ' + formatDate(d, 'MM-dd-yy');
// Today Is: 02/21/07
```

1.6 Protecting a Viewsheet from Script Errors

Because scripts are executed when the Viewsheet is generated on the server, a script error can cause Viewsheet generation to fail. To prevent this from happening, you should wrap error-prone code inside a "try-catch" block, which allows you to trap errors before they affect Viewsheet execution. See Appendix JS.10.4, *The 'try-catch' Statement*, for more information.

See Also
Debugging a Viewsheet Script, for information on locating errors.

1.7 Debugging a Viewsheet Script

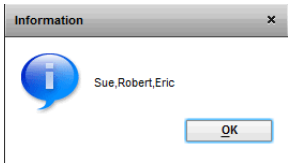
When you attempt to debug a Viewsheet script, it is often helpful to view the current values of variables and objects. To do this, use the `alert()` function, which opens a dialog box to display a specified string.

```
alert('string to display')
```

```
Note: The alert() function does not pause script execution.
```

For example:

```
var arr = ['Sue','Robert','Eric'];
alert(arr.join(","))
```



You can also use the `alert()` dialog box to display critical information to a user (e.g., that the user's selections have resulted in an empty dataset being returned).

See Also

Protecting a Viewsheet from Script Errors, to improve Viewsheet robustness.

2 Adding Script to a Viewsheet

There are three levels at which you can attach script in a Viewsheet:

1. **Property level**: Attach script as a property expression to dynamically modify property values (e.g., visibility, color, etc.) of the host component. See *Adding Property Script (Expressions)*.

2. **Component level**: Attach script to an individual component (Chart, Gauge, etc.) to modify component properties, including properties that cannot be set by an expression, or to implement other logic. See *Adding Component Script*.

3. **Viewsheet level**: Attach script in one of two global domains, the *onInit Handler* or the *onRefresh Handler*. See *Adding Viewsheet Script* for more information.

2.1 Script Evaluation Order

Unlike conventional JavaScript on web pages, JavaScript that you embed in a Viewsheet is executed as part of the Viewsheet generation process on the server, *not* the client browser. However, scripts can control client-side interactions through event handlers and hyperlinks.

The overall sequence of Viewsheet script evaluation is as follows.

1. Execute the *onInit Handler*.

2. Execute the *onRefresh Handler*.

3. Execute all component-level scripts. (See *Adding Component Script*.)

The *onInit Handler* executes only once, when the Viewsheet is first loaded. It can be used to initialize variables and define parameters. The *onRefresh Handler* re-executes each time the Viewsheet is refreshed by a user operation.

Component scripts and property expressions are re-executed whenever the particular component is refreshed due to a user operation (on the component itself, or on a related component). Because the order in which individual component scripts are initially executed is not determined, you should avoid setting properties of one element from within a script attached to a different element.

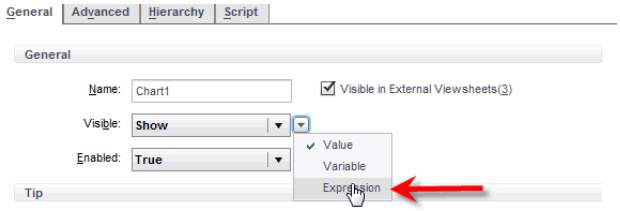
2.2 Accessing Java in Script

You can call both JavaScript and Java functions from within Viewsheet scripts. However, scripts are executed in a restricted environment that provides access only to the following Java packages: `java.awt`, `java.text`, `java.util`, `java.sql`, `inetsoft.graph.*`, `inetsoft.report`, `inetsoft.report.lens`, `inetsoft.report.filter`, `inetsoft.uql`, and `inetsoft.report.painter`.

To permit scripts to access Java packages other than those listed, an administrator can set the `javascript.java.packages` property in the *sree.properties* file to specify additional packages. To add package contents to the current namespace, use `importPackage('packageName')`. For example, `importPackage('java.lang')` or `importPackage(java.lang)` allows scripts to refer to object names without full qualification (i.e., '`String`' rather than '`java.lang.String`').

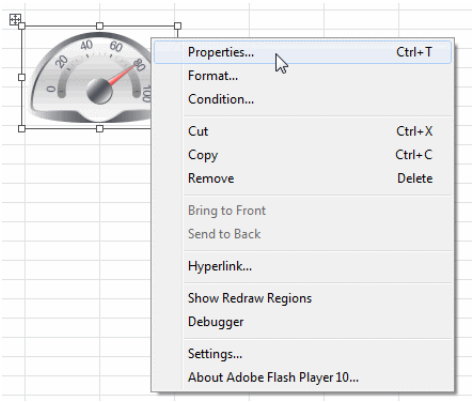
2.3 Adding Property Script (Expressions)

You can enter a script in any property field that provides an 'Expression' option.



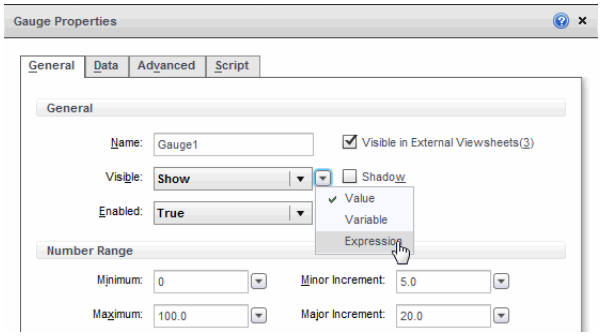
For example, to add a script to a component expression field, follow the steps below:

1. Right-click the component, and select 'Properties' from the context menu. This opens the 'Properties' dialog box.

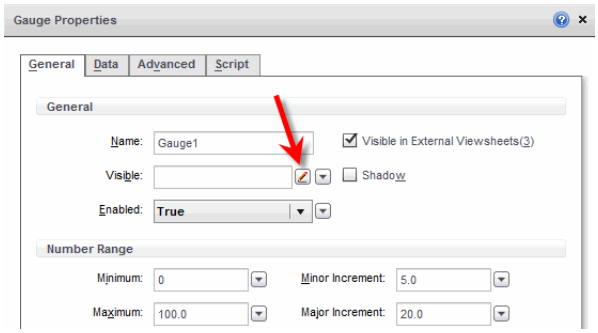


2. In the ‘Properties’ dialog box, find the property you wish to dynamically change. (In the example, we use the ‘Visible’ property under the **General** tab.)

3. Click the arrow button next to the property you want to modify, and select ‘Expression’ from the menu. This enables a small ‘Edit’ button.



4. Click the ‘Edit’ button. This opens the Formula Editor.

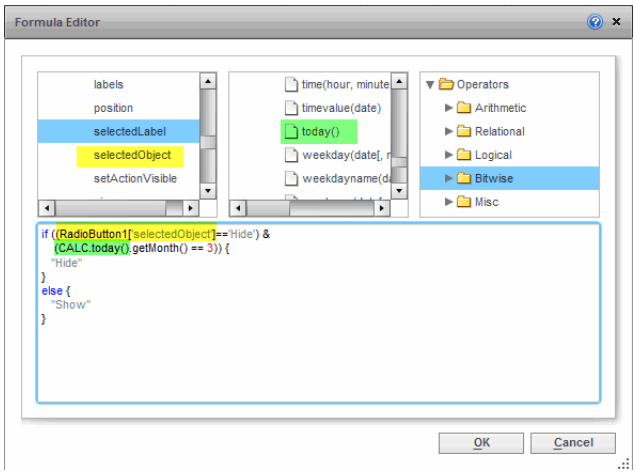


Note: The script must return one of the options in the property menu. For example, return values for the ‘Visible’ property must be one of “Show,” “Hide,” or “Hide on Print and Export.” (To achieve “Show on Print and Export” behavior, use the special expression “!visible”.)

5. In the Formula Editor, enter the desired script. For example, the following script hides the component if the user has selected ‘Hide’ from a RadioButton control, and the current month is March (i.e., the third month).

```
if ((RadioButton1['selectedObject']=='Hide') &
    (CALC.today().getMonth() == 3)) {
   "Hide"
}
else {
   "Show"
}
```

To automatically insert a function (such as `CALC.today()`), click the corresponding option in the ‘Functions’ list above the edit area. To automatically insert a reference to another component’s property, click the corresponding option in the ‘Component’ list above the edit area



6. Click ‘OK’ to close the Formula Editor, and click ‘OK’ to close the ‘Properties’ dialog box.

Note: An expression script can reference properties of its hosting component without full qualification. E.g., for a script on component “SelectionList1”, 'SelectionList1.selectedObject' and 'selectedObject' refer to the same property.

Expressions must always return permissible values for the corresponding property. For example, if a property requires an integer, an expression that sets that property must evaluate to an integer. If a property requires one of a fixed set of options, e.g., {‘Rows’, ‘Columns’}, then the expression that sets that property must evaluate identically to one of those strings (‘Rows’ or ‘Columns’).

See Also

Dynamic Properties, in *Advanced Dashboard Design*, to set properties with Input components.

Matching Dynamic Values to Property Values, in *Advanced Dashboard Design*, for required return values.

Introduction to JavaScript Programming, for information on JavaScript programming.

2.4 Adding Component Script

Note: Text element script does not re-execute when the user makes Viewsheet selections. To re-execute script that modifies a Text element, place the script in the *onRefresh Handler*.

A script attached to a Viewsheet component is evaluated whenever the component is refreshed, which includes initial Viewsheet loading, and user interactions with the component. The script can change the properties of the element, including visibility.

Properties set in script override properties set in the component ‘Properties’ dialog box.

In fact, a component script can access any component in the Viewsheet. However, it is not good practice to modify the properties of one element from within the script of another element. See *Debugging a Viewsheet Script* for alternative approaches.

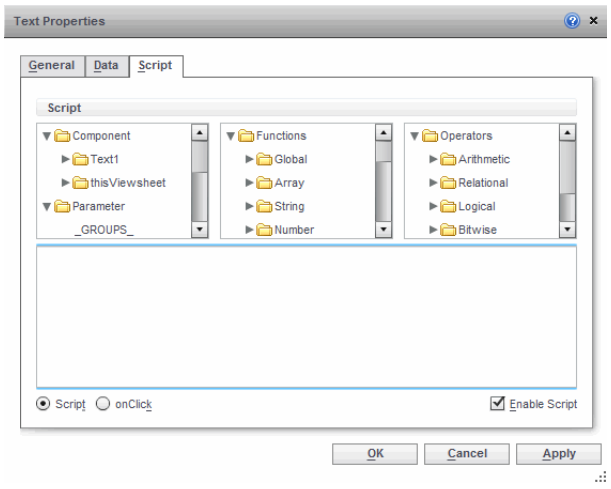
To attach a script to a Viewsheet component, follow the steps below.

1. Right-click the desired component, and select ‘Properties’ from the context menu. This opens the ‘Properties’ dialog box.

2. Click the **Script** tab to open the Script Editor.

```
Do not attach an onClick script to a component that already has a
hyperlink.
```

3. For a Text component, select the **Script** or **onClick** radio button at the bottom of the dialog box.




```
Note: A script can reference properties of its hosting component
without full qualification. E.g., for a script on component
“SelectionList1”, 'SelectionList1.selectedObject' and
'selectedObject' refer to the same property.
```

4. Type the desired script in the text field, and press ‘OK’.

To temporarily disable a script (without deleting it entirely), uncheck the ‘Enable Script’ option at the bottom right corner of the Script Editor.

See Also
Script Evaluation Order, for the order of script evaluation.

2.4.1 Common Component Properties

All elements share a basic set of properties, including foreground, background, font, alignment, visibility, etc. The following sections explain how to use these properties.

Color Property

Color properties (foreground/text and background/fill) are frequently used to highlight a text or textbox element. The simplest way to specify a color property is to assign a string containing one of the constants from the java.awt.Color class: `black`, `blue`, `cyan`, `darkGray`, `gray`, `green`, `lightGray`, `magenta`, `orange`, `pink`, `red`, `white`, and `yellow`.

```
foreground = 'red';
```

`Since color is a Java type, the class name must be fully qualified.`

You can also specify a color as java.awt.Color object, an integer (e.g., hexadecimal) representing the RGB value of a color, an array of RGB values, or a JSON object.

```
foreground = java.awt.Color.red;
background = 0xFF0000; // RRGGBB
foreground = [255, 0, 0];
foreground = {r:255,g:0,b:0};
```

Alternatively, you can create a color object by calling the constructor with the 'new' operator.

```
foreground = new java.awt.Color(0.5, 1, 0);
```

Note that the parameters to the color constructor have type float. Because JavaScript treats all numbers as float by default, you have to explicitly convert them to integer if you want to specify the RGB values in the range of 0-255. The default float parameters pass the RGB value in the range of 0-1, where 1 is equivalent to 255 in the integer version.

Font Property

You can specify the font property with a string containing the font name, style, and size, separated by dashes, or by creating a java.awt.Font object.

```
font = 'Verdana-BOLD-12';
font = new java.awt.Font('Verdana', java.awt.Font.BOLD, 12);
```

The name of the font can be a TrueType font name, or a logical font name. Logical font names are not recommended, however, because the logical font may be replaced by a different font in the runtime environment.

There are three font styles, `Font.PLAIN`, `Font.BOLD` and `Font.ITALIC`. The styles can be combined with a bitwise OR.

```
font = new java.awt.Font('Verdana', java.awt.Font.BOLD |
java.awt.Font.ITALIC, 12);
```

The final parameter specifies the size of the font. Style Intelligence provides an extended font that supports additional styles:

Table 4. Font Styles

FONT STYLE	DESCRIPTION
Underline	Draw an underline below the text. The line style can be any one of the Style Intelligence *Line Styles*.
Strikethrough	Draw a line through the text in the middle.
Superscript	Draw the text at the upper corner of the previous text.
Subscript	Draw the text at the lower corner of the previous text.
Smallcaps	Draw all letters in capital letter, but draw the lowercase letters in a smaller size.
Allcaps	Convert all letters to uppercase.
Shadow	Draw the text with a shadow effect.

To create an extended font, you must use the fully qualified name of `inetsoft.report.StyleFont` class.

```
font = new inetsoft.report.StyleFont('Verdana',
         java.awt.Font.BOLD |
         inetsoft.report.StyleFont.UNDERLINE, 12,
         StyleConstant.THIN_LINE);
```

The final parameter specifies the line style used to draw the underline.

See Also

Appendix JS.13, *StyleConstant Object*, for commonly-used constant values.

Visibility Property

You can dynamically hide or show an element using script. Typically, you would test a condition or user input at runtime, and then hide one or more elements to present an appropriate view.

```
if(field['discount'] == 0) {
  Text8.visible = false;
}
```

As for all element properties, you should change the visibility of an element only from within a script attached to the *same* element, or from within the *onRefresh Handler*. Otherwise, the location of the element in the report may influence the outcome.

See Also

Debugging a Viewsheet Script, for information on making scripts self-contained.

Script Evaluation Order, for information about order of evaluation and its effects.

Alignment Property

To align text within Text elements, use the Report Designer alignment controls.

The Alignment property controls both the horizontal and vertical alignment of report elements in the page layout. The horizontal alignment is specified by the alignment constants `H_LEFT`, `H_CENTER`, and `H_RIGHT`. (See *Alignments*.)

```
alignment = StyleConstant.H_CENTER;
```

The vertical alignment property can be combined with the horizontal alignment property by using the bitwise OR.

```
alignment = StyleConstant.H_CENTER |
StyleConstant.V_CENTER;
```

The four vertical alignment options are `V_TOP`, `V_CENTER`, `V_BOTTOM`, and `V_BASELINE`.

See Also

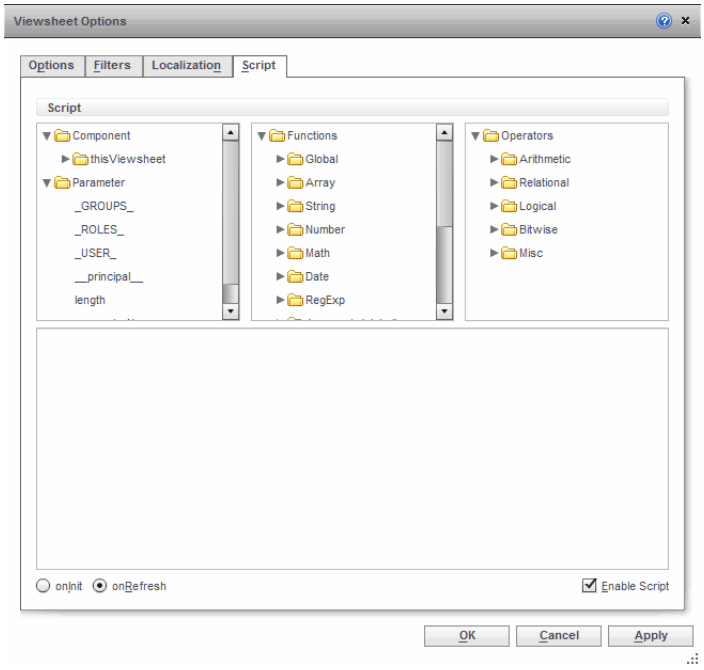
Appendix JS.13, *StyleConstant Object*, for commonly-used constant values.

Text, for more information about Text properties.

2.5 Adding Viewsheet Script

To create a global Viewsheet script, follow the steps below:

1. Click ‘Options’ button in the toolbar to open the ‘Viewsheet Options’ dialog box.
2. Click the **Script** tab. This opens the Script Editor.



3. Using the radio buttons at the bottom, select either **onInit** or **onRefresh**.

4. Enter the desired script, and press 'OK'.

To temporarily disable a script (without deleting it entirely), uncheck the 'Enable Script' option at the bottom right corner of the Script Editor.

The following sections provide additional information about the *onInit Handler* and *onRefresh Handler*.

See Also
Adding Script to a Viewsheet, for information about scripting elements.

2.5.1 onInit Handler

A Viewsheet's *onInit* script is executed only once, when the Viewsheet it first loaded. It is ideal for one-time initialization tasks, such as defining Viewsheet parameters and variables.

A variable declared in *onInit* script will have global scope, and be accessible to individual component scripts. To declare a local variable with the same name elsewhere, use the '`var`' keyword in the declaration to remove ambiguity.

See Also
onRefresh Handler, to declare report-level variables, or validate user parameters.

2.5.2 onRefresh Handler

The onRefresh script does not execute when you refresh a Preview by clicking the **Preview** tab. To run the onRefresh script, create a new Preview. (See *Previewing a Viewsheet* in *Advanced Dashboard Design*.)

The *onRefresh* handler is similar to the *onInit* handler, and is executed following *onInit* upon Viewsheet loading. However, unlike *onInit*, an *onRefresh* script is executed every time the Viewsheet is refreshed. Because the Viewsheet is refreshed in response to most user interactions, including selections, the *onRefresh* handler gives you the ability to dynamically modify multiple elements from a central location.

A variable declared in *onRefresh* script will have global scope, and will be accessible to individual component scripts. To declare a local variable with the same name elsewhere, use the 'var' keyword in the declaration to remove ambiguity.

See Also

The Chart bindingInfo Property, for information on data binding for Charts.

onInit Handler, to declare session-level variables or report parameters.

3 Accessing User Input in Script

Tabs have a property 'selectedIndex' which returns the numeric representation of the active tab.

You can access the state of different components within the Viewsheet using the `selectedObject`, `selectedObjects`, and `drillMember` keywords. See *Walkthrough: Accessing User Selections* for an example using `selectedObject`. See *Walkthrough: Interactive Chart Drilldown* for an example using `drillMember`.

- *selectedObject*, *selectedLabel*: The selected value or label of the component: Applies to single value input elements (Sliders, Radio Buttons, etc.) and the tabs of co-located components. For example, to return the value or label selected in a radio button control:

```
radioButton.selectedObject
radioButton.selectedLabel
```

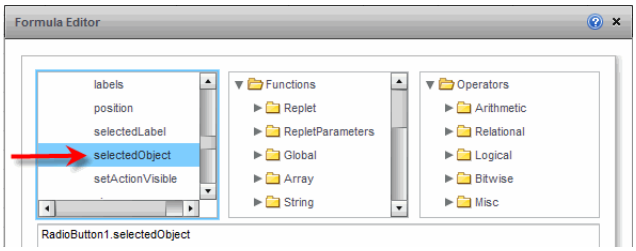
- *selectedObjects*, *selectedLabels*: An array of values or labels selected in a component: Applies to multi-value input elements (e.g., Check Boxes) and selection elements (Selection Lists and Trees). For example, to return the first selected item or label in a Selection List:

```
SelectionList.selectedObjects[0];
SelectionList.selectedLabels[0];
```

- *drillMember*: For Selection Trees, the name of the field that resides one level in the hierarchy below the user's lowest selected level:

```
SelectionTree.drillMember;
```

You can manually type these references into your script, or enter them automatically by clicking the appropriate node on the 'Component' tree of the Script Editor.



See Also

Accessing User-Modified Data in a Table, for information on accessing user table edits.

Accessing the Most Recent User Modification, for information on accessing the last user edit.

3.1 Walkthrough: Accessing User Selections

Walkthrough

This example uses a RadioButton to give the user a choice of two input options, a Selection List, and a Range Slider. Follow the steps below:

1. Create a new Viewsheet based on the ‘ProductSales’ Worksheet in the Tutorial folder.

2. Add a table, Selection List, and Range Slider. To do this, follow the steps below:

    a. Drag the ‘SalesByDate’ Data Block into the Viewsheet. This creates a new table called ‘SalesByDate’. Resize the table as desired.

    b. From the ‘SalesByDate’ Data Block in the Component tree, drag the ‘State’ column to an empty region on the Viewsheet. This creates a new Selection List named ‘State’.

    c. From the Component tree, drag a Range Slider control into the Viewsheet. This creates a new Range Slider.

    d. From the ‘SalesByDate’ Data Block in the Component tree, drag the ‘Date’ column onto the Range Slider. This binds the Range Slider to the Date field.

3. Create a control to allow the user to select the input method. Follow the steps below:

    a. From the Component tree, drag a Radio Button control into the Viewsheet. This creates a new Radio Button.

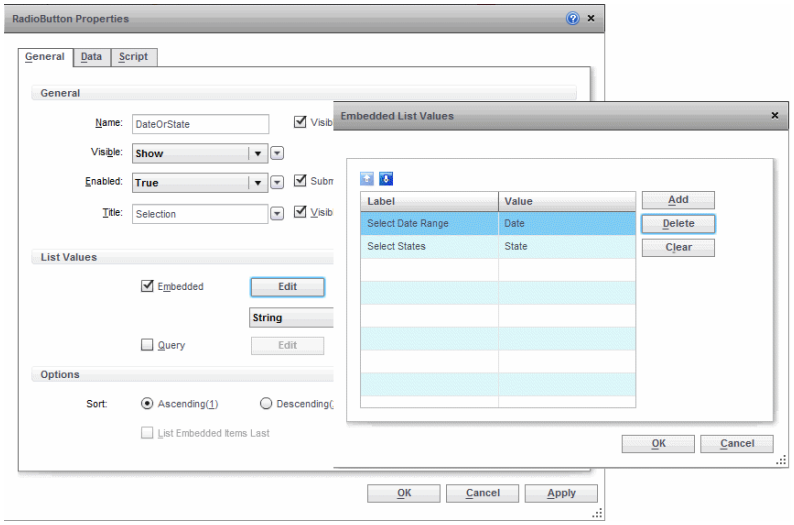
    b. Right-click the RadioButton, and select ‘Properties’ from the context menu. This opens the ‘Properties’ dialog box.

    c. In the RadioButton ‘Properties’ dialog box, enter the text “DateOrState” in the ‘Name’ field. Enter the text ‘Selection’ in the ‘Title’ field.

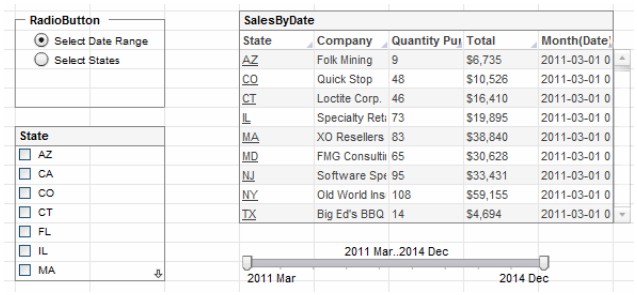
    d. In the ‘List Values’ panel of the ‘Properties’ dialog box, select ‘Embedded’ and click the ‘Edit’ button. This opens the ‘Embedded List Values’ dialog box.

    e. Click ‘Add’. For the first ‘Label’ enter the text “Select Date Range.” For the corresponding ‘Value’, enter the text “Date”.

    f. Click ‘Add’ again. For the second ‘Label’ enter the text “Select States.” For the corresponding ‘Value’, enter the text “State”.



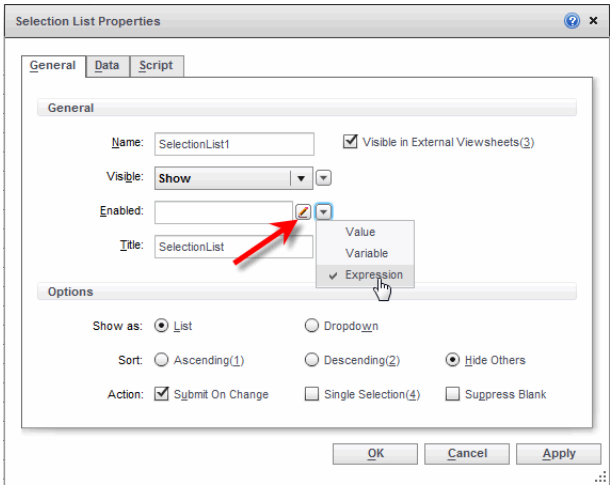
g. Click ‘OK’ in both dialog boxes. This creates a Radio Button with two choices from which the user can select the input method.



4. Add expression scripts to the Selection List and the Range Slider so that the appropriate control is enabled for the given user selection. Follow the steps below:

   a. Right-click the Selection List, and select ‘Properties’ from the context menu to open the ‘Properties’ dialog box.

   b. For the ‘Enabled’ option, click the right-side pop-up menu, and select ‘Expression’. Then click the ‘Edit’ icon to open the Formula Editor.



c. In the Formula Editor for the Selection List, enter the following script:

```
= if (DateOrState.selectedObject=='Date') {
   false
}
else {
   true
}
```

This indicates that the Selection List control should be disabled whenever ‘Date’ is selected in the “DateOrState” RadioButton, and enabled otherwise.

d. Click ‘OK’ twice to exit the Formula Editor and the ‘Properties’ dialog box.

e. Repeat the above steps for the Range Slider, and set the ‘Enabled’ script as follows:

```
= if (DateOrState.selectedObject=='Date') {
   true
}
else {
   false
}
```

f. Click ‘OK’ twice to exit the Formula Editor and the ‘Properties’ dialog box.

5. Preview the Viewsheet, and experiment with the controls.

You can hide the control completely by adding similar scripts (returning values 'Show' and 'Hide') to the 'Visibility' property.

Note that when you choose 'Select Date Range', only the Range Slider is enabled, and when you choose 'Select States', only the Selection List is enabled.

3.2 Walkthrough: Interactive Chart Drilldown

Walkthrough

In this example, you will create a Selection Tree that filters the geographical regions displayed on a Chart. You will use the *drillMember* property of the Selection Tree to dynamically adapt the Chart X-axis to display the appropriate geographical level (Region, State, or City), based on user selections.

Follow the steps below:

1. Create a new Viewsheet based on the 'Sales' > 'Sales Explore' Worksheet.

2. From the **Component** panel, drag a Chart component into the Viewsheet grid. Resize the Chart as desired.

3. From the **Component** panel, drag a Selection Tree component into the Viewsheet grid.

4. Add a three-level geographical hierarchy to the Selection Tree. Follow the steps below:

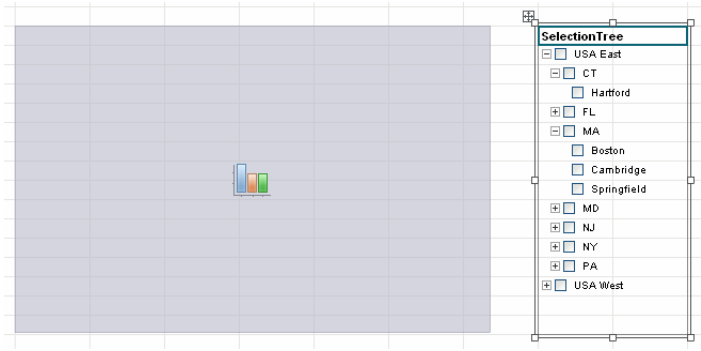
    a. Expand the 'Sales' Data Block in the **Component** panel.

    b. Drag the 'Region' field from the Data Block onto the Selection Tree. This adds the 'Region' data to the tree.

    c. Drag the 'State' field from the Data Block onto the Selection Tree. This adds the 'State' data to the tree.

    d. Drag the 'City' field from the Data Block onto the Selection Tree. This adds the 'City' data to the tree.

    e. Resize the Selection Tree so that it can display a reasonable number of options.



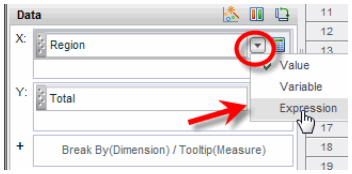
5. Press the ‘Edit’ button on the Chart to open the Chart Editor.

6. From the **Data Source** panel of the Chart Editor, drag the ‘Total’ field (Under ‘Measures’) to the ‘Y’ region in the **Data** panel.

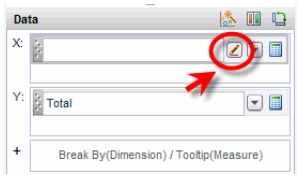
7. From the **Data Source** panel of the Chart Editor, drag the ‘Region’ field (Under ‘Dimensions’) to the ‘X’ region in the **Data** panel.

8. Configure the Chart to adapt the X-axis binding based on the user’s selection in the Selection Tree. To do this, follow the steps below:

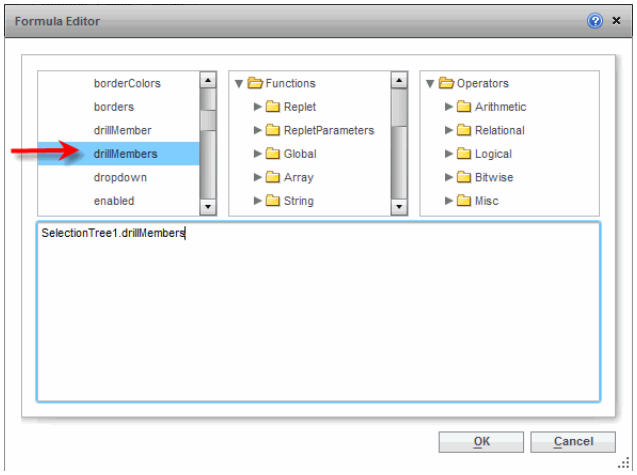
   a. Press the “arrow” button next to the ‘Region’ dimension in the **Data** panel. In the pop-up menu, select the ‘Expression’ option.



   b. Press the ‘Edit’ button to open the Script Editor.

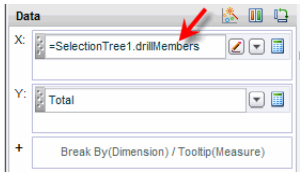


   c. In the top-left panel of the Script Editor, expand the ‘`SelectionTree1`’ node, and click ‘`drillMember`’. This adds the `SelectionTree1.drillMember` property in the editing area.



The *drillMember* property returns the name of the field that resides one level (in the Selection Tree hierarchy) below the *lowest* level that the user has selected. Thus, if the user has selected one or more *states* in the Selection Tree, `SelectionTree1.drillMember` returns the string '`City`'.

d. Press 'OK' in the Chart Editor. The output of `SelectionTree1.drillMember` is now assigned to the Chart X-axis binding.



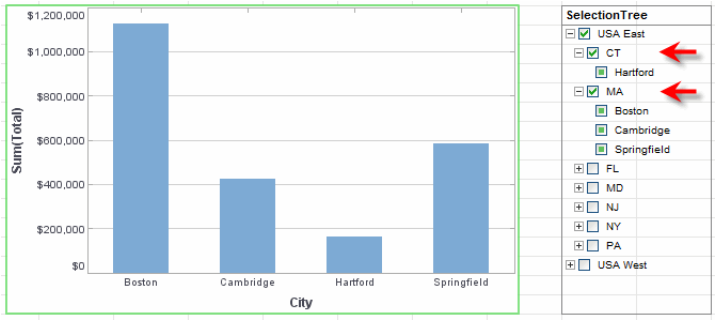
9. Verify that the X-axis binding adapts to the user's selections:

a. Press the 'Clear Selection' button in the Selection Tree to remove any existing selections. Observe that the Chart X-axis now represents the 'Region' dimension, the top level of the Selection Tree hierarchy.

b. Select the 'USA East' option in the Selection Tree. Observe that the Selection Tree now filters the Chart to display only data from the 'USA East' region. In addition, the Selection Tree `drillMember` property changes the X-axis binding to represent the 'State' dimension, which is one level below 'Region' in the Selection Tree hierarchy.

c. Select the 'CT' and 'MA' options in the Selection Tree. Observe that the Selection Tree now filters the Chart to display only data from the states of 'CT' and 'MA'. In addition, the Selection Tree

`drillMember` property changes the X-axis binding to represent the ‘City’ dimension, which is one level below ‘State’ in the Selection Tree hierarchy.



See Also

drillMembers, to show multiple geographical levels simultaneously on the map.

Drilling Down into a Chart, for information on the built-in drill-down feature.

4 Accessing Component Data

For Table, Crosstab, and Selection List components, expression scripts can acquire actual data values from the component using relative or absolute references.

4.1 Accessing Table Data

There are two key properties for accessing the values in a table, *table* and *data*.

- `table` – A two-dimensional array containing the table data *as displayed*. The array includes header rows as well as data rows.
- `data` – A two dimensional array containing the *raw* table data (prior to grouping and summarization). It does not include header rows.

Two sub-properties that are especially useful when looping through the rows or columns of tables are '`length`' and '`size`'.

```
Note: A table that returns no data still displays the column header
row. Therefore, table.length is 1 in the no-data case.
```

- *`table.length`*/*`table.size`* – The number of rows and columns (respectively) in the table, *as displayed*, including column header row.
- *`data.length`*/*`data.size`* – The number of rows and columns (respectively) in the *original* table (prior to grouping and summarization), including column header row.

As an example, consider the following table script, which iterates through all data rows (beginning with row index 1, the first data row) and columns of a table, and cumulatively sums these values.

```
var tot = 0;
for(var row = 1; row < table.length; row++) {
   for(var col = 0; col < table.size; col++) {
      tot = tot + table[row][col];
   }
}
```

See Also

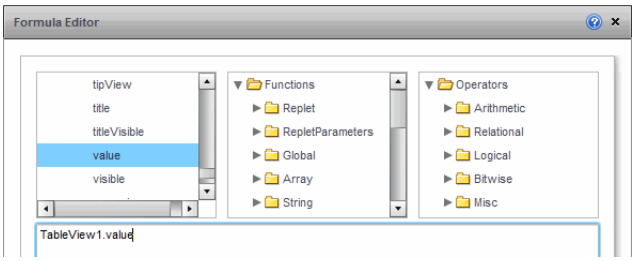
Accessing User-Modified Data in a Table, for information on how to access an editable table.

4.2 Accessing Data With Relative References

The keywords below allow you to access cells within a table or Selection List by using relative indexing (i.e., reference in relation to current cell).

SYNTAX	DESCRIPTION
`field['col_name']`	Value in the current row of column named "col_name."
`row`	Row index of the current cell.
`col`	Column index of the current cell.
`data[rowIx][colIx]`	Value of the data at index [rowIx,colIx] in the raw data (before grouping or summarization). Indices may use keywords 'row' and 'col' to create relative references.
`table[rowIx][colIx]`	Value of the data at [rowIx,colIx] in the table (as displayed). Indices may use keywords 'row' and 'col' to create relative references.
`value`	Value of the current cell.

You can type these keywords manually into your script, or you can click the appropriate node on the 'Component' tree in the Script Editor.



The following sections illustrate how to access data using these keywords.

See Also

Accessing Data With Absolute/Mixed References, for information on absolute references.

4.2.1 Accessing Data in the Current Cell ('value')

The `value` keyword refers to the value in the current cell. The `value` keyword is the equivalent of the index expression `table[row][col]`.

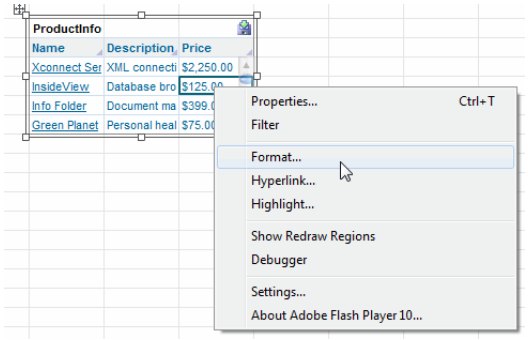
Walkthrough

This example sets the text colors and fill colors of table cells based on data values within the table. If a price is greater than $1000, the price displays in red text, otherwise, in blue text.

1. Create a Viewsheet based on the 'Tutorial' > 'ProductSales' Data Worksheet.

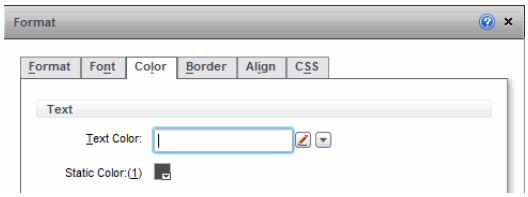
2. Drag and drop the ‘ProductInfo’ Data Block from the Component pane into the Viewsheet. Expand the table so that you can see more rows.

3. Select any data cell under the ‘Price’ column. Right-click and choose the ‘Format’ option. This opens the ‘Format’ dialog box.



4. Select the **Color** tab.

5. From the ‘Text Color’ menu select the ‘Expression’ option, and click the ‘Edit’ button. This opens the Formula Editor.



6. Enter the following script in the Formula Editor:

```
if (value > 1000) {
   [255,0,0];        // red
} else {
   0x0000FF;         // blue
}
```

Note: A color can be specified as an ordered triple, or as a hexadecimal number.

7. Click ‘OK’ to exit the Formula Editor, and then click ‘OK’ to exit the ‘Format’ dialog box.

Note: You can apply conditional formatting without using script. See *Highlights* in the *Advanced Dashboard Design*.



ProductInfo		
Name	Description	Price
Xconnect Serve	XML connectivity tool	$2,250.00
InsideView	Database browser	$125.00
Info Folder	Document management t	$399.00
Green Planet	Personal health monitor	$75.00
House Hunter	House hunting assistant	$129.00
Fast Mail	Lightweight email agent	$39.00
Sync Me	Palm power sync	$145.00
WebCalendar	Web enabled calendar/o	$248.00
True Action	Java game toolkit	$398.00
Fancy Menus	Collection of menus	$49.00
Mega Icons	Over 10,000 icons	$550.00
MeToo AppServ	Application server	$3,000.00
Easy Chess	Chess game	$35.00

Notice that the cells in the ‘Price’ column are colored red if the value is greater than 1000 and blue otherwise. The script affects the entire ‘Price’ column, and can be edited from the ‘Format’ dialog box of any cell in the column. This example continues in *Accessing Data in the Same Row (‘field’)*.

See Also
value, for reference information.

4.2.2 Accessing Data in the Same Row (‘field’)

You can access the value in the current row of another column by using the `field` keyword. For example, `field['col_name']` refers to the value in the current row of a column named “col_name.”

Walkthrough

This example continues from *Accessing Data in the Current Cell (‘value’)*. It will set the color of the ‘Name’ column based on the value of the ‘Price’ column:

1. Select a cell in the ‘Name’ column. Right-click and choose ‘Format’.

2. Select the **Color** tab. From the ‘Text Color’ menu select the ‘Expression’ option, and click the ‘Edit’ button. This opens the Formula Editor.

3. In the Formula Editor, enter the following expression:

```
if (field['Price'] > 1000) {
   [255,0,0];
} else {
   [0,0,255];
}
```

4. Click ‘OK’ to exit the Formula Editor, and then click ‘OK’ to exit the ‘Format’ dialog box.



ProductInfo		
Name	Description	Price
Xconnect Serve	XML connecti	$2,250.00
InsideView	Database bro	$125.00
Info Folder	Document ma	$399.00
Green Planet	Personal heal	$75.00
House Hunter	House huntin	$129.00
Fast Mail	Lightweight e	$39.00
Sync Me	Palm power s	$145.00
WebCalendar	Web enabled	$248.00
True Action	Java game to	$398.00
Fancy Menus	Collection of	$49.00
Mega Icons	Over 10,000 i	$550.00
MeToo AppServ	Application s	$3,000.00
Easy Chess	Chess game	$35.00

The values in the ‘Name’ column are now highlighted based on the corresponding values in the ‘Price’ column. This example continues in *Accessing Data in Different Row or Column (‘row’/‘col’)*.

See Also
field, for reference information.

4.2.3 Accessing Data in Different Row or Column (‘row’/‘col’)

To access any cell in a table by relative reference, use the indexing expression `table[rowIx][colIx]` or `data[rowIx][colIx]`.

Predefined variables `row` and `col` allow you to refer to the row and column of the current cell, that is, the cell whose format is being modified. You can use these variables to construct the desired offsets from the current cell.

For example:

- `table[row][col]`: Value in current cell (same as `value` keyword).
- `table[row][col-1]`: Value in current row, one column to left.
- `table[row-1][col]`: Value in current column, one row above.
- `table[row+1][col]`: Value in current column, one row below.

The `data` and `table` constructs yield the same results for tables, but may produce different results when used in charts/crosstab tables. The `data` syntax references the data prior to any summarization, while the `table` syntax references the tabular data as displayed.

Walkthrough

This examples continues from *Accessing Data in the Same Row (‘field’)*. It highlights every price that is less than the price immediately above it in the table, and uses the `row` and `col` variables to create relative references:

1. Select any data cell under the ‘Price’ column. Right-click and choose the ‘Format’ option.

2. Select the **Color** tab. From the 'Fill Color' menu select the 'Expression' option, and click the 'Edit' button. This opens the Formula Editor.

3. In the Formula Editor, enter the following script:

```
if (row > 1) {
  if (value < table[row-1][col]){
    [255,255,0];
  }
}
```

4. Click 'OK' to exit the Formula Editor, and then click 'OK' to exit the 'Format' dialog box.



ProductInfo		
Name	Description	Price
Xconnect Server	XML connectivity tool	$2,250.00
InsideView	Database browser	$125.00
Info Folder	Document manageme	$399.00
Green Planet	Personal health monit	$75.00
House Hunter	House hunting assista	$129.00
Fast Mail	Lightweight email age	$39.00
Sync Me	Palm power sync	$145.00
WebCalendar	Web enabled calenda	$248.00
True Action	Java game toolkit	$398.00
Fancy Menus	Collection of menus	$49.00
Mega Icons	Over 10,000 icons	$550.00
MeToo AppServer	Application server	$3,000.00
Easy Chess	Chess game	$35.00

The cells in which the value is less than the previous row's price are now highlighted in yellow.

See Also

row, *col*, for reference information.

4.3 Accessing Data With Absolute/Mixed References

The following expressions allow you to access any cell in a table by absolute reference.

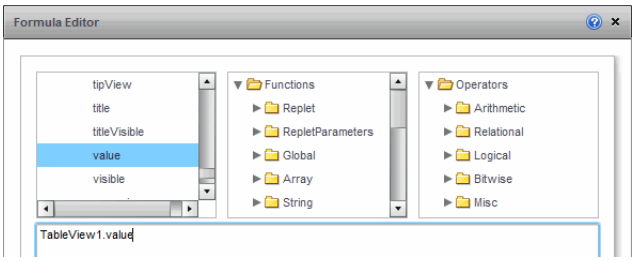
```
table[rowIx][colIx]
data[rowIx][colIx]
```

Simply set the `rowIx` and `colIx` indexes to specify the position of the desired cell. For example:

- `table[0][0]`: Value in first row, first column.
- `table[2][4]`: Value in third row, fifth column.

The "`table.length`" property yields the total number of rows in the table. The `data` and `table` constructs yield the same results for tables, but may produce different results when used in crosstab tables. The `data` syntax references the data prior to any summarization, while the `table` syntax references the tabular data as displayed.

You can type these keywords manually into your script, or you can click the appropriate node on the 'Component' tree in the Script Editor.



You can use absolute and relative references in the same statement, as the following example illustrates.

Example: Using Absolute and Relative References

This example continues from *Accessing Data in Different Row or Column ('row'/'col')*. It will set the header cell of the 'Price' column to show a red background if there exists a price exceeding $2000, and will otherwise show a green background. This requires a loop structure, as well as mixed absolute and relative references:

1. Select the header cell in the 'Price' column. Right-click and choose 'Format'.

2. Select the **Color** tab. From the 'Fill Color' menu select the 'Expression' option, and click the 'Edit' button. This opens the Formula Editor.

3. Add the following script in the Formula Editor.

```
for (i=1; i<table.length; i++) {
   if (table[i][col] > 2000) {
      [255,0,0];
      break;
   }
   else {
      [0,255,0];
   }
}
```

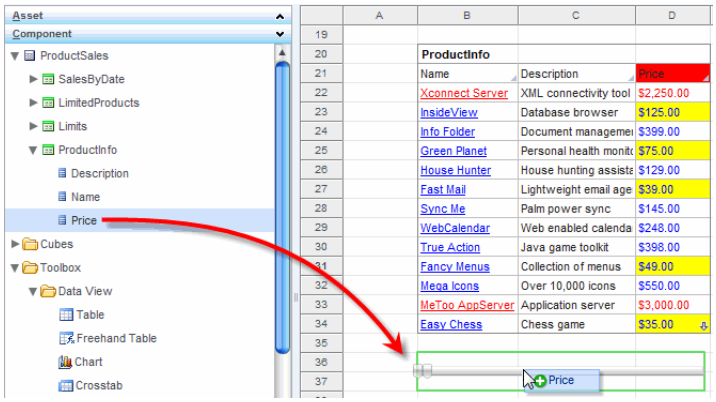
4. Click 'OK' to exit the Formula Editor, and then click 'OK' to exit the 'Format' dialog box.



The header cell is red because a price exists that exceeds $2000. To test the header cell script, add a Range Slider to filter the values in the table. Follow these steps:

5. Drag a Range Slider from the Component tree into the Viewsheet. This creates a new Range Slider.

6. From the 'ProductInfo' Data Block in the Component tree, drag the 'Price' field onto the Range Slider. This binds the Range Slider to the 'Price' field.



7. Drag the right side of the Range Slider to vary the prices included in the dataset. Observe the color of the header cell as the dataset changes.

¢

See Also

Accessing Data With Relative References, for information on relative references.

4.4 Accessing User-Modified Data in a Table

When you configure a Viewsheet table to be "editable" (see *Table Properties: Advanced Tab*), this allows users to insert, delete, or modify

data in the table. Changes that the user makes to the table, however, do not persist after the Viewsheet is closed.

To allow user modifications to persist beyond the current session, you must commit the modified records to the database. (The following section, *Committing User-Modified Data to Database (Database Write-Back)*, provides a full example.) The `getFormRows` table function provides you with records that have been modified.

- `Table1.getFormRows(ADDED)`: Provides a two-dimensional array containing the new rows that have been inserted by the user. The length of the array is equal to the number of inserted rows, and the width of the array is equal to the number of columns in the table.
- `Table1.getFormRows(CHANGED)`: Provides a two-dimensional array containing the rows that have been modified by the user. The length of the array is equal to the number of modified rows, and the width of the array is equal to the number of columns in the table.
- `Table1.getFormRows(DELETED)`: Provides a two-dimensional array containing the rows that have been deleted by the user. The length of the array is equal to the number of deleted rows, and the width of the array is equal to the number of columns in the table.
- `Table1.getFormRow(idx)`: Provides the row designated by index `idx`.

See *Adding Component Script* for information about the onClick handler.

These functions are typically called within the onClick handler of a *Submit Button*, as shown in the following section.

When a user adds or changes a row in an editable table, the new or modified row is highlighted in color. To clear the highlight on these rows and remove them from the corresponding array returned by `getFormRows`, use the table's *commit(type)* function.

- `Table1.commit(ADDED)`: Clears highlights on added rows, and removes them from array returned by `getFormRows(ADDED)`.
- `Table1.commit(CHANGED)`: Clears highlight on changed rows, and removes them from array returned by `getFormRows(CHANGED)`.
- `Table1.commit(DELETED)`: Removes deleted rows from array returned by `getFormRows(DELETED)`.
- `Table1.commit(idx)`: Removes the row returned by `getFormRows(idx)` from the array of modified rows.

See Also

Accessing Table Data, for information on how to access data in a regular table.

Accessing the Most Recent User Modification, to retrieve a value that the user has just entered.

getFormRow(row), for function reference information.
getFormRows([type]), for function reference information.
Table Properties: Advanced Tab, in *Advanced Dashboard Design*, for information on setting a table as editable.
Validating User Input, in *Advanced Dashboard Design*, for information on restricting the values that a user can enter.
Committing User-Modified Data to Database (Database Write-Back), for information on updating the database with modified information.

4.5 Accessing User-Modified Data from TextInput

To acquire the data that a user has entered in a *TextInput* component, simply reference the component's `value` property:

```
TextInput1.value
```

To commit this acquired data to the database, see *Committing User-Modified Data to Database (Database Write-Back)*.

See Also
Accessing the Most Recent User Modification, to retrieve a value that the user has just entered.
Validating User Input, in *Advanced Dashboard Design*, for information on restricting values a user can enter.
Committing User-Modified Data to Database (Database Write-Back), for information on updating the database with modified information.

4.6 Accessing the Most Recent User Modification

When a user makes a modification to an editable table or Input component, the change triggers the Viewsheet's *onRefresh Handler* script to re-execute. The onRefresh handler offers an '*event*' object which you can use to detect the user's action and make an appropriate response. The *event* object provides access to the following Viewsheet events:

- Name and region of the Editable Table in which the user has just made a modification.
- Name and type of an Input Component in which the user has just made a selection.

For an editable table, the '*event*' object provides the following properties:

name	The name of the table component.
type	The type of the component, always 'table'.
row	The index of the modified row in the table.
column	The index of the modified column in the table.
source	A reference to the table object which can be used to access table properties.

Example: Recalculating Table Data

The following example illustrates how you can use the '*event*' object to dynamically update table data in response to user modifications. You will create a table that contains the fields 'Name', 'Price', 'Quantity', and 'Total'. You will enable the user to modify the 'Price' and 'Quantity' fields, and use script to automatically update the 'Total' field with the corresponding data (i.e., the product of 'Price' and 'Quantity').

To create this example, follow the steps below:

1. Create a new Viewsheet based on the 'Order Model' data model. To do this, follow the steps below:

   a. Press the 'New Viewsheet' button in the toolbar. This opens the 'New Viewsheet' dialog box.

   b. Expand the 'Data Source' node, and then expand the 'Orders' node.

   c. Select the 'Order Model' entry, and press 'OK'. This creates a new Viewsheet based on the 'Order Model' data model.

2. From the Viewsheet **Component** panel, drag a Table component to the Viewsheet grid.

3. Expand the 'Order Model' node at the top of the **Component** panel, and drag the following fields into the Table component:

   ```
   Product.Name
   Product.Price
   Product.Quantity Purchased
   Product.Total
   ```

4. Resize the table as desired.



Table

Product:Name	Product:Price	Product:Quan	Product:Total
19 inch LCD	$310.00	14	$4,340
House Hunter	$129.00	11	$1,419
Fast Mail	$39.00	17	$663
NetStorage	$2,850.00	6	$17,100
MeToo AppServe	$3,000.00	2	$6,000
Animal World	$59.00	2	$118
Barbie's Fashion	$79.00	12	$948
Wireless Mouse	$40.00	1	$40
Animal World	$59.00	4	$236
Learn ABC	$75.00	13	$975
Web Bridge	$105.00	7	$735
MeToo AppServe	$3,000.00	2	$6,000

5. Right-click the table and select 'Properties'. This opens the 'Table Properties' dialog box.

6. Under the **Advanced** tab, select 'Enable Table Editing' and press 'OK'.

7. In the table, right-click the ‘Product:Name’ header and select ‘Column Option’ from the context menu.

8. Deselect the ‘Enable Column Editing’ option, and press ‘OK’. This prevents the user from editing the product name.

9. In the table, right-click the ‘Product:Total’ header and select ‘Column Option’ from the context menu.

10. Deselect the ‘Enable Column Editing’ option, and press ‘OK’. This prevents the user from manually editing the total. (The total will be calculated by script.)

11. Press the ‘Options’ button in the Viewsheet toolbar. This opens the ‘Viewsheet Options’ dialog box.

12. Select the **Script** tab. Press the ‘onRefresh’ radio button to select the onRefresh handler.

13. Enter the following script into the Editor:

```
if(event != null) {
  var tableObject = event.source;
  var row = event.row;
  var column = event.column;

  if(column == 1 || column == 2) {
    var editedRow = tableObject.getFormRow(row);
    var price = editedRow[1]; // second column = Price
    var quant = editedRow[2]; // third column = Quantity

    if(price != null && quant != null) {
      tableObject.setObject(row, 3, price*quant); // fourth
column = Total
    }
    else {
      tableObject.setObject(row, 3,'');
    }
  }
}
```

    Observe that `event.row` and `event.column` are used together with `getFormRow` to acquire the particular value that the user has entered, and `setObject` is used to write the calculated data into the table.

14. Press ‘OK’ to close the Script Editor.

15. Preview the Viewsheet.

Edit various values in the ‘Price’ and ‘Quantity Purchased’ columns, and observe how the ‘Total’ column is automatically updated. Note, however, that changes a user makes to the table (including calculated values) do not persist across sessions unless they are committed to the database. See

Committing User-Modified Data to Database (Database Write-Back) for more details.

¢

See Also

Table Properties: Advanced Tab, in *Advanced Dashboard Design*, for information on how to make a table editable.

4.7 Committing User-Modified Data to Database (Database Write-Back)

The previous section, *Accessing User-Modified Data in a Table*, described how you can acquire the records in a table which have been modified (inserted, deleted, or changed) by the user. If you want the user's changes to persist across sessions, you must commit these changes to the database. This is known as *database write-back*.

In most cases, you should commit records to the database by placing a script within the onClick handler of a *Submit Button*. The script should perform the following operations:

1. Create a connection to the database:

   ```
   var conn =
   createConnection('datasource','username','password');
   ```

   If the specified data source does not require credentials, replace the username and password with empty strings (`''`).

2. Prepare the desired SQL statement, using placeholders for values to be inserted.

   ```
   var pstmt = conn.prepareStatement("insert into table1 values
   (?,?,?)");
   ```

   The quoted string should be a valid SQL statement which will perform the desired database operation when values are inserted each placeholder ("?") in the next step.

See *Accessing User-Modified Data in a Table* for more information about the getFormRows() function.

3. Insert values for SQL placeholders. In most cases, these values will be taken from the arrays returned by the table functions `getFormRows(ADDED)`, `getFormRows(CHANGED)`, and `getFormRows(DELETED)`. For example:

   ```
   pstmt.setInt(1,rowsAdded[i][0]);     // first substitution
   pstmt.setString(2,rowsAdded[i][1]);  // second
   substitution
   pstmt.setString(3,rowsAdded[i][2]);  // third substitution
   ```

The integer parameter in each of the above functions is a reference to the corresponding placeholder ("?") in the SQL statement; for example, the parameter value "1" indicates a substitution for the first "?", "2" indicates a substitution for the second "?", and "3" indicates a substitution for the third "?". The row of the array is indexed by variable "i" above because the array is generally accessed iteratively within a loop structure.

4. Update the database tables by executing the SQL statement:

   ```
   pstmt.executeUpdate();
   ```

   Note that this does not yet commit any changes to the database. Changes are not committed until the 'commit' function is called (next step).

5. Commit the changes:

   ```
   conn.commit();
   ```

6. If necessary (in case of an error), roll back the changes:

   ```
   conn.rollback();
   ```

7. Close the connection to the data source:

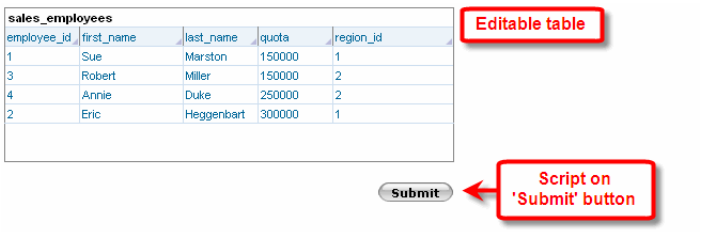
   ```
   conn.close();
   ```

The *Submit Button* onClick script below provides a complete example, which updates a database table with rows which have been added, deleted, and changed.

Example: Committing User Edits to Database

The following sample Viewsheet, based on the `sales_employees` table in the 'Orders' database, contains an editable table with five columns. The Viewsheet table, also named `sales_employees`, contains all of the columns from `sales_employees` table in the Orders data source.

User editing is enabled only for the `first_name`, `last_name`, and `quota` columns. (See *Table Properties: Advanced Tab* and *Validating User Input* in *Advanced Dashboard Design* for details on configuring table and column editing features.)



Note: The ‘Orders’ data source used in this example is a Microsoft Access database, different than the ‘Orders’ data source installed by default. The default Derby database is not writable, so it cannot be used for this example.

The following script acquires the rows that were added, deleted, or changed, and then commits the modifications back to the database. The script is placed in the onClick handler of the Submit button.

```
try{
  // Create the connection to data source (no credentials)
  var conn=createConnection('Orders','','');

  // Find new rows added by user
  var rowsAdded = sales_employees.getFormRows(ADDED);

  // Find existing rows changed by user
  var rowsChanged = sales_employees.getFormRows(CHANGED);

  // Find existing rows deleted by user
  var rowsDeleted = sales_employees.getFormRows(DELETED);

  //---------- ADDING ROWS ----------//
  if(rowsAdded!=null){
    // Iterate through array of added rows
    for(var i = 0; i<rowsAdded.length; i++){

      // Prepare SQL statement with placeholders
      var pstmt=conn.prepareStatement("insert into
sales_employees values (?,?,?,?,?)");

      // Insert placeholder values from ith row of array
      pstmt.setInt(1,rowsAdded[i][0]); // row i, column 1
      pstmt.setString(2,rowsAdded[i][1]); // row i, column
2
      pstmt.setString(3,rowsAdded[i][2]); // row i, column
3
      pstmt.setInt(4,rowsAdded[i][3]); // row i, column 4
      pstmt.setInt(5,rowsAdded[i][4]); // row i, column 5

      // Execute the SQL statement
      pstmt.executeUpdate();

      // Commit the changes
      conn.commit();
    }
  }

  //---------- DELETING ROWS ----------//
  if(rowsDeleted!=null){
    // Iterate through array of deleted rows
    for(var i = 0; i<rowsDeleted.length; i++){

      // Prepare SQL statement with placeholders
      var pstmt = conn.prepareStatement("delete from
sales_employees where employee_id=?");

      // Insert placeholder value from ith row of array
      pstmt.setInt(1,rowsDeleted[i][0]); // row i, column 1

      // Execute the SQL statement
      pstmt.executeUpdate();
```

```
      // Commit the changes
      conn.commit();
    }
  }

  //---------- CHANGING ROWS ----------//
  if(rowsChanged!=null){
    // Iterate through array of changed rows
    for(var i = 0; i<rowsChanged.length; i++){

      // Prepare SQL statement with placeholders
      var pstmt = conn.prepareStatement("update
sales_employees set first_name=?, last_name=?, quota=?
where employee_id=?");

      // Insert placeholder values from ith row of array
      pstmt.setString(1,rowsChanged[i][1]); // row i,
column 2
      pstmt.setString(2,rowsChanged[i][2]); // row i,
column 3
      pstmt.setInt(3,rowsChanged[i][3]); // row i, column 4
      pstmt.setInt(4,rowsChanged[i][0]); // row i, column 1

      // Execute the SQL statement
      pstmt.executeUpdate();

      // Commit the changes
      conn.commit();
    }
  }
}

// In case of error, roll-back changes.
catch(e) {
  conn.rollback();
}

// Close the data source connection
finally{
  conn.close();
}
```

¢

5 Accessing Worksheet Data

In both chart and property scripting, you have explicit access to the data contained in the Worksheet Data Block. The following sections explain the syntax for accessing the Worksheet data.

See Also

Running a Query from Script, for information on how to access data from an arbitrary query or Worksheet.

Adding Property Script (Expressions), for information on setting component properties.

Adding Component Script, for information on manipulating charts with script.

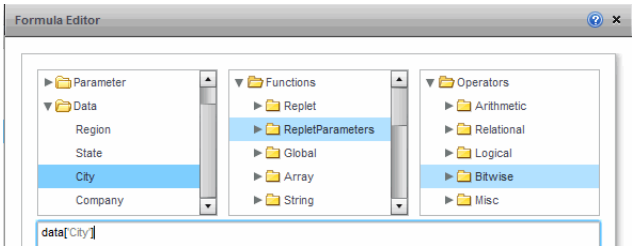
5.1 Referencing Data in the Worksheet Data Block

Within a Viewsheet script, you can use the standard Worksheet syntax to access data from any Worksheet Data Block within the linked Worksheet. For example, to access a value in a Worksheet Table or Embedded Table called “TopSales”, use one of the following syntaxes.

```
Note: In general, Data Block operations should be performed in the
Worksheet rather than the Viewsheet.
```

- `TopSales[2][4]`:
  Third row, fifth column of the Data Block “TopSales.”
- `TopSales[2]['Price']`:
  Third row of the "Price" field in the Data Block “TopSales.”
- `viewsheet["Top Sales"][2][4]`:
  Third row, fifth column of the Data Block “TopSales” (with space in name).

You can type these keywords manually into your script, or you can insert them by clicking the appropriate node on the ‘Data’ tree in the Script Editor.



See Also

Grouping, in *Advanced Data Mashup*, for information on aggregating data.

Accessing Table Cells in Script, in *Advanced Data Mashup*, for information on referencing Data Blocks.

5.2 Aggregating Data in the Worksheet Data Block

To aggregate data from the Worksheet's Data Block within Viewsheet script, use the following syntax:

```
aggregateMethod(DataBlockName['ColumnName'])
```

```
Note: In general, Data Block operations should be performed in the
Worksheet rather than the Viewsheet.
```

For example, to obtain the sum or average of the Data Block column 'Price', use the following expressions:

```
var sumPrice = Sum(TopSales['Price'])
var avePrice = Average(TopSales['Price'])
```

To reference a Data Block that has a space in the name (e.g., "Top Sales"), use the `viewsheet` keyword as follows:

```
var avePrice = Average(viewsheet["Top Sales"]['Price'])
```

See Also

Grouping, in *Advanced Data Mashup*, for information on aggregating data.
Accessing Table Cells in Script, in *Advanced Data Mashup*, for information on referencing Data Blocks.

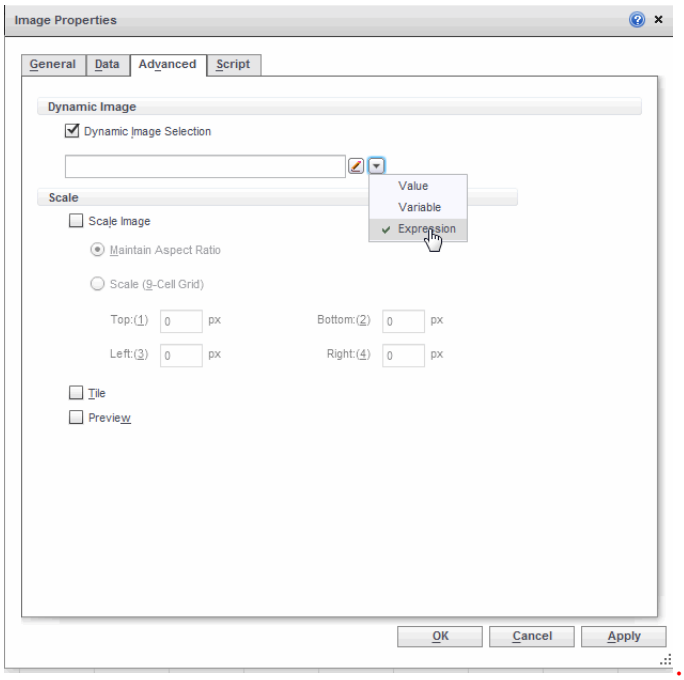
6 Changing Images with Script

You can dynamically set the image file (GIF, JPG, etc.) of an image component using a variable or an expression. Follow the steps below:

1. Add an image element into the Viewsheet. Right-click and select the ‘Properties’ option. This opens the ‘Image Properties’ dialog box.

2. Select the **Advanced** tab and select ‘Dynamic Image Selection’.

3. From the drop-down menu, select ‘Expression’ or ‘Variable’, depending on the mechanism you wish to use to set the image property.



4. If you choose ‘Expression’, click the ‘Formula Editor’ button, and add a script in the text area. A typical script might take the following form:

```
if(condition) {
   'image1.gif';
} else {
   'image2.gif';
}
```

You can specify an image by the uploaded image name (e.g., `'image1.gif'`), a resource path on the server (e.g., `'/library/`

`image1.gif'`), or a URL (e.g., "`http://www.google.com/images/logos/ps_logo2.png`").

5. Click 'OK' to close the Formula Editor, and then click the 'OK' to close the 'Image Properties' dialog box.

The Image component now displays the specified image file.

See Also

Image, for scripting functions related to Images.
getImage(string), for another approach to setting images dynamically.
Creating an Image, in *Advanced Dashboard Design*, for information on uploading and referencing images.

7 Running a Query from Script

Note: runQuery does not retrieve data from Data Models.

To run a query defined in the query registry (*query.xml*) or any available Data Worksheet, use the *runQuery(query_name [,parameters])* command.

```
// Run a query:
var q = runQuery('total sales');

// Run the query for primary Data Block in a Worksheet:
var q = runQuery('ws:global:worksheetName');

// Run the query for non-primary Data Block in a Worksheet:
var q = runQuery('ws:global:worksheetName:tableName');
```

For Data Worksheets, qualify the name as 'ws:global:path' or 'ws:user_name:path' for global or user scope, respectively.

In either case, the `runQuery` results are returned as a two-dimensional array, whose first row contains the column headers. You can access the query values using standard array indexing.

```
// Assign data in first column, first row to Text
component:
Text1.value = q[1][0];
```

Query parameters (if any) can be passed as part of the `runQuery` call. For example, to pass the Viewsheet '`start_time`' parameter to the query as '`start_time`' and pass the current date/time as '`end_time`':

```
var q = runQuery('total sales', [['start_time',
parameter.start_time],['end_time', new Date()]]);
```

See Also

Binding Data to a Chart in Script, for various approaches to binding data to a chart.

runQuery(query_name [,parameters]), for additional information on runQuery.

Accessing Worksheet Data, to access data in the underlying Worksheet.

Advanced Data Mashup, for information on creating Worksheet assets.

Appendix VS: Viewsheet Object Reference

This appendix describes all objects in the Viewsheet environment.

See Also
Common Reference, for general JavaScript and charting functions.
Report Scripting, for functions specific to the reporting environment.

VS.1 Global Functions and Properties

The Viewsheet environment provides a few global functions and properties in addition to those defined by the standard JavaScript runtime (see Appendix JS: *General JavaScript Functions*). This section presents the additional functions.

VS.1.1 thisViewsheet

This section presents global Viewsheet functions that address Viewsheet appearance and scheduling.

thisViewsheet.ADDACTION(ICON,LABEL,ACTION)

Adds a button to the Viewsheet toolbar to perform a scripted action.

See *Localization/Internationalization* in *Advanced Dashboard Design* for more information about the SreeBundle file.

Parameters

```
icon          The file containing the button image (21x21),
a resource, file path, or URL (see below)
label         A string specifying the button label or an ID
that maps to a label in the SreeBundle file
action        A Viewsheet script that executes when the
button is pressed (e.g., the name of a function to execute)
```

Example

```
thisViewsheet.addAction('dateButton.png','Get
date...',"alert(formatDate(CALC.today(),'MMMM dd, yyyy'))")
```

The `icon` parameter can specify a file name, resource path, file path, or URL. If specified as a resource, the image must be located on the classpath. If the `icon` parameter specifies a simple file name, the server will search for the file in the following locations:

- The image directory specified on the 'Presentation' > 'Viewsheet' > 'Toolbar Icons' page of Enterprise Manager. (See *Viewsheet Toolbar Icons* in *Advanced Server Administration*.)
- The directory *webapps\sree\WEB-INF\classes\inetsoft\fl\images*, if the Flash version of the Viewsheet is displayed.
- The directory *webapps\sree\WEB-INF\classes\inetsoft\html\images*, if the HTML version of the Viewsheet is displayed.

The button image should be square with sides of length 21 pixels. If the server cannot find the image, a default button image will be substituted.

thisViewsheet.SCHEDULEACTION(BOOLEAN[,EMAILS])

Specifies whether the Viewsheet should be executed as part of a scheduled task. Set to `true` to allow the Viewsheet to run as scheduled. Set to `false` to override the scheduled task setting and suppress Viewsheet execution.

Parameters

Boolean	true (default): execute task false: do not execute task
emails	Array of emails

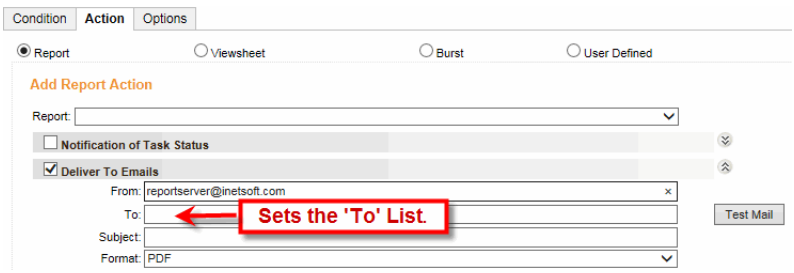
You can use the `scheduleAction` function to conditionalize the execution of a scheduled Viewsheet task based on actual Viewsheet data. For example, the following script uses the value of a Gauge component to determine whether the Viewsheet task should execute as scheduled.

Example

```
if (Gauge1.value > 10000) {
  thisViewsheet.scheduleAction(true)
}
else {
  thisViewsheet.scheduleAction(false)
}
```

If the scheduled action is a user notification (for example), then the user will receive the alert only when the value of the Gauge component exceeds the threshold.

Use the optional second parameter, `emails`, to dynamically set the list of emails for the 'Deliver To Emails' scheduled action. The list that you provide in the `emails` array supersedes any emails specified on the Scheduler **Action** tab (both for user-scheduled tasks and administrator-scheduled tasks), and automatically enables the 'Deliver To Emails' task action.



Example

```
if (Gauge1.value > 10000) {
  emails = [joe@inetsoft.com, sue@inetsoft.com,
mark@inetsoft.com];
```

```
}
else {
  emails = [joe@inetsoft.com];
}
thisViewsheet.scheduleAction(true, emails);
```

See Also

Setting an Alert, in *Using the Portal*, for end-user alert specification.
Setting an Alert, in *Advanced Server Administration*, to set an alert based on highlight or range condition.
Scheduling a Dashboard, for information on Scheduling Viewsheets.

thisViewsheet.ISACTIONVISIBLE(NAME)

Returns the visibility setting for the specified toolbar button.

Parameters

name	name of toolbar button (String): 'Home' 'Previous' 'Next' 'Refresh' 'Add Bookmark' 'Browse Bookmarks' 'Email' 'Print' 'Export' 'Edit' 'Toolbar'

Example

```
if (!thisViewsheet.isActionVisible("Export")) {
  alert('Some features are disabled.');
}
```

thisViewsheet.SETACTIONVISIBLE(NAME,BOOLEAN)

Sets the visibility of Viewsheet toolbar buttons. To hide a button, set the value to `false`.

Note: The setActionVisible property for *individual* components is documented with the other component properties.

Parameters

name	name of toolbar button (String): 'Home' 'Previous' 'Next' 'Refresh' 'Add Bookmark' 'Browse Bookmarks' 'Email' 'Print' 'Export' 'Edit' 'Toolbar'
boolean	false: hide the specified button true: show the specified button

Example

```
thisViewsheet.setActionVisible("Export", false);
```

thisViewsheet.UPDATETIME

Returns the date/time at which data was most recently obtained by the Viewsheet.

- If a materialized view is in effect for the Viewsheet (see *Materialized Views* in *Advanced Server Administration*), `thisViewsheet.updateTime` returns the date/time at which the most recent materialized view was generated for the Viewsheet.
- If a materialized view is *not* in effect, `thisViewsheet.updateTime` returns the date/time of the most recent query execution for the Viewsheet.

Type

Date	Date/time at which data was obtained

Example

```
Text1.text = 'Last Updated at ' +
formatDate(thisViewsheet.updateTime,'MMM dd, h:mm a');
```

See Also

Materialized Views, in *Advanced Server Administration*, for information about scheduling materialized views.

VS.1.2 parameter

The `parameter` property allows you to read and write Variables that have been defined in a Worksheet (*Creating a Variable* in *Advanced Data Mashup*) or Input controls that have been defined in a Viewsheet. The syntax is

```
parameter.variableName
```

where `variableName` is the name of the Variable asset defined in the Worksheet or the Input component defined in the Viewsheet.

```
Example: parameter.stateSelector
Example: parameter.RadioButton1
```

You can also define new parameters within the Viewsheet itself.

```
Example: parameter.myParamName = 'Hello';
```

See Also

Input Components, in *Advanced Dashboard Design*, for more information on Input controls.

Creating a Variable, in *Advanced Data Mashup*, for more on using Variable assets.

parameter._GROUPS_

Returns an array of groups to which the current user belongs.

Type

Array of Strings	Groups to which currennt user belongs

Example

```
alert('User belongs to: ' + parameter._GROUPS_.join(', '))
```

parameter._ROLES_

Returns an array of roles possessed by the current user.

Type

Array of Strings	Roles possesed by current user

Example

```
alert('User roles: ' + parameter._ROLES_.join(', '))
```

parameter._USER_

Returns the user name of the current user.

Type

String	User name of current user

Example

```
alert('Logged in as ' + parameter._USER_)
```

parameter.__PRINCIPLE__

Returns the SRPrincipal object, which contains session information.

Type

Object	SRPrincipal object

Example

```
var p = parameter.__principal__;
p.setParameter('State', ['NJ', 'NY', 'CT']);
```

See Also

Accessing the User Session with SRPrincipal, in *Advanced Server Integration*, for details on writing user data to SRPrincipal.

parameter.LENGTH

Returns the number of currently defined parameters.

Type

Integer	Number of currently defined parameters

Example

```
alert(parameter.length)
```

parameter.PARAMETERNAMES

An array containing the current parameter names.

Type

String	Parameter Name

Example

```
alert(parameter.parameterNames[0])
```

VS.1.3 Special Functions

The next sections describe some special purpose functions.

PVIEWSHEET

For a Viewsheet that is accessed via drilldown (hyperlink or auto-drill) from another Viewsheet, the `pviewsheet` property provides access to the current settings of components within the parent Viewsheet.

For example, if the parent Viewsheet contains a Selection List named 'Company', you can obtain the array of currently selected companies in the 'Company' Selection List within the drilldown Viewsheet script by using the following syntax.

```
pviewsheet.Company.selectedObjects
```

See Also

Viewsheet Object Reference, for a list of accessible Viewsheet component properties.
Hyperlinks, in *Advanced Dashboard Design*, for information on Viewsheet drilldowns.
Auto-Drilldown, in *Advanced Data Modeling*, for information on configuring data-level hyperlinks.
pviewsheet in *Report Scripting*, to access component settings from within a drilldown report.

EVENT

User interactions with editable tables, Input components, and Selection components automatically re-execute the Viewsheet onRefresh handler.

The *event* object can be used within an *onRefresh Handler* script to provide information about the user's most recent action in editable *Tables*, *Input Components*, and *Selection Components* (Selection List, Selection Tree, Range Slider, Calendar). The *event* object is instantiated at runtime when a user makes an interaction of the appropriate type, so you should test for its existence before attempting to access its properties. For example:

```
if(event!=null) {
```

```
   // access 'event' properties, such as...
   // event.name, event.type, event.source, etc.
}
```

For an editable Table, the *event* object provides the following properties:

event.name	The name of the table component.
event.type	The type of the component, always 'table'.
event.row	The index of the row in the table which the user modified.
event.column	The index of the column in the table which the user modified.
event.source	A reference to the table object which can be used to access table properties.

For an Input component or Selection component, the *event* object provides the following properties:

event.name	The name of the component, e.g., "SelectionList1"
event.type	The type of the component: 'radiobutton', 'checkbox', 'combobox', 'textinput', 'selectionlist', 'selectiontree', 'rangeslider', 'calendar'
event.source	A reference to the component which can be used to access component properties, e.g., `event.source.title='New Title'`.

See Also

Accessing the Most Recent User Modification, for additional discussion of '*event*' object.

RUNQUERY(QUERY_NAME [,PARAMETERS])

Returns a result set as a two-dimensional array that can be used within Viewsheet script.

Parameters

query_name a string containing the query name
parameters (optional) a two dimensional string array, each row containing a name-value pair that corresponds to a query parameter

Example

```
var rs =
runQuery('orders',[['category','Business'],['price',100]]);
```

See Also

Running a Query from Script, for examples of executing Worksheet queries.

GETIMAGE(STRING)

Loads an image from a Java resource, URL, or database BLOB (binary large object). The `getImage()` function checks whether the argument

specifies a valid resource. If it does not specify a resource, then `getImage()` treats the parameter as an encoded image. Both ASCII Hex and ASCII85 are supported (useful for XML data).

To retrieve a BLOB from a database, first use a Data Worksheet to create a data block with a column bound to the image data field. Then use `getImage()` to reference the desired data block cell, as shown in the example below.

Parameters

```
string        image file resource path or URL
              or
              data block cell containing BLOB data
              (ascii hex/ascii85 gif/jpeg image data)
```

Example

```
// resource path
Image1.image = getImage('/com/mypackage/icon.gif');
```

Example

```
// URL
Image1.image = getImage('http://visualizefree.com/images/
inetsoft.png');
```

Example

```
// 'picture' column in data block 'Query1' contains BLOBs
Image1.image = getImage(Query1[1]['picture']);
```

See Also

Changing Images with Script, for another approach to dynamically setting images.

ALERT(MESSAGE)

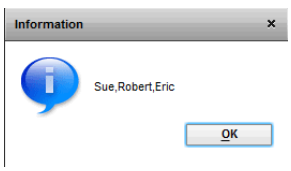
Opens a dialog box to display a specified message. This function does not pause script execution.

Parameters

```
message       a string containing the message to display
```

Example

```
var arr = ['Sue','Robert','Eric'];
alert(arr.join(","))
```



See Also

confirm(message), to prompt the user with a choice dialog box.

CONFIRM(MESSAGE)

See *Adding Component Script* for information about the onClick handler.

The `confirm()` function opens a dialog box to provide the user with a choice to continue or cancel an operation. For example, you can use the `confirm()` function in the onClick script of a 'Submit' button to verify the user's intension to submit.

The `confirm` function does not suspend script processing. Rather, a script containing the `confirm` function is automatically executed either once or twice:

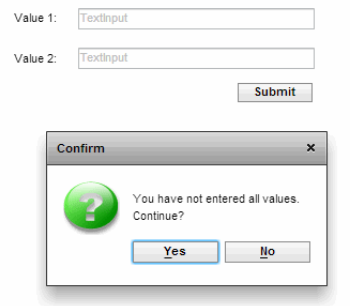
1. On the first execution of the script, the property `confirmEvent.confirmed` is set to `false`.

2. On the second execution of the script, which occurs only if the user presses 'Yes' in the 'Confirm' dialog box, the property `confirmEvent.confirmed` is set to `true`.

This repeated script execution allows you to invoke different operations depending on the user's response to the confirmation prompt. For example, the script below asks the user to confirm that they want to continue submission even though they have not provided values for both TextInput components.

```
if(!confirmEvent.confirmed) {
  if((TextInput1.value==null) || (TextInput2.value==null))
  {
    confirm('You have not entered all values.  Continue?')
  }
}
else {
  parameter.val1 = TextInput1.value;
  parameter.val2 = TextInput2.value;
  alert('Input has been submitted.')
}
```

The script executes a first time when a user presses the 'Submit' button. During this first execution, `confirmEvent.confirmed` is `false`, so the TextInput values are tested and the 'Confirm' dialog box is generated.



If the user presses 'No', the dialog box is closed and no action is taken. If the user presses 'Yes', the script executes a second time with `confirmEvent.confirmed` set to `true`, which commences the action in the `else` block. (In this case, the desired action is simply to assign the TextInput values to corresponding Viewsheet parameters.)

See Also
alert(message), to display a dialog box message to the user.
Submit Button, in *Advanced Dashboard Design*, for information on how to create a Submit button.

VS.2 Common Properties

This section describes basic properties that are shared by most of the Viewsheet components. Exceptions are noted where appropriate.

ALIGNMENT

Specifies the horizontal and vertical alignment of an element. Its value is a bitwise OR of a horizontal alignment option and a vertical alignment option.

HORIZONTAL ALIGNMENT	VERTICAL ALIGNMENT
StyleConstant.H_LEFT	StyleConstant.V_TOP
StyleConstant.H_CENTER	StyleConstant.V_CENTER
StyleConstant.H_RIGHT	StyleConstant.V_BOTTOM
StyleConstant.H_CURRENCY	StyleConstant.V_BASELINE

Type
`integer`

Example
```
alignment = StyleConstant.H_CENTER | StyleConstant.V_CENTER;
```

See Also
Alignment Tab, for more information about alignments.

ALPHA

Specifies the transparency/opacity of the component's fill color as a value between 0 (completely transparent) and 100 (completely opaque).

Type
`Number`

Example
```
alpha = 35.5;
```

See Also
Color Tab, for more information about setting color.

BACKGROUND

Specifies the background color that fills the area of the element.

Type

```
java.awt.Color          e.g., java.awt.Color.BLUE
number (hex)            e.g., 0xFF0000
string (color name)     e.g., 'red'
array [r,g,b]           e.g., [255,0,0]
JSON {r:#,g:#,b:#}      e.g., {r:255,g:0,b:0}
```

Example

```
background = java.awt.Color.lightGray;
```

See Also

Color Tab, for more information about setting color.

BORDERCOLORS

Specifies the element border colors as an array of four values in the following order: `[top,bottom,left,right]`. Each color can be of one of the types given below.

Type

```
java.awt.Color          e.g., java.awt.Color.BLUE
number (hex)            e.g., 0xFF0000
string (color name)     e.g., 'red'
array [r,g,b]           e.g., [255,0,0]
JSON {r:#,g:#,b:#}      e.g., {r:255,g:0,b:0}
```

Example

```
borderColors = [[255,0,0], 0x444444, 'blue', 'black'];
```

See Also

Color Tab, for more information about setting color.

BORDERS

Specifies the individual line styles for each border of the element, as an array of form `[top,left,bottom,right]`.

Type

```
Array of Line Styles
```

Example

```
borders =
[StyleConstant.DOUBLE_LINE,StyleConstant.NO_BORDER,
StyleConstant.THIN_LINE, StyleConstant.NO_BORDER];
```

See Also

Border Tab, for additional information on borders.

DATACONDITIONS

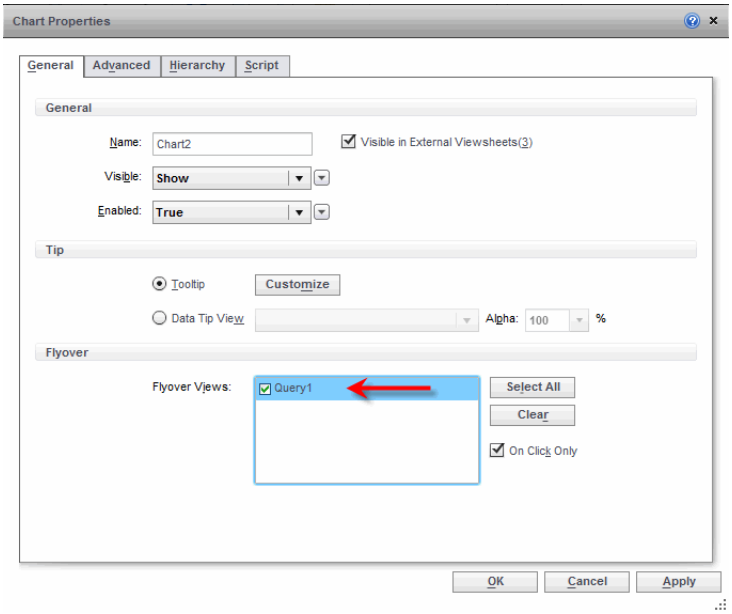
Returns the *Flyover View* conditions that are currently in effect on a filtered component. The `dataConditions` property provides an array of attributes, `attr[]`, and an array of values, `value[]`, that contain the conditions for all flyover filters currently being applied to a specified component.

Type

`dataConditions.attr[]` `labels`	`Array of filtered dimension`
`dataConditions.value[]` `values`	`Array of filtered dimension`

Example

Consider a Chart that uses a flyover view to filter a Table named “Query1”. If the ‘On Click Only’ option is set for the Chart flyover (and therefore multiple Chart groups can be selected in the flyover), the Table’s `dataConditions.attr[]` and `dataConditions.value[]` arrays will each contain one item for every group the user selects on the Chart. You can iterate through these arrays to obtain the currently selected dimensions and values.



This script could also be written as an expression for the ‘Title’ property in the **General** tab.

The script below is placed in the **Script** tab of the Table component (“Query1”) that is being filtered, and modifies the ‘`title`’ property of the Table.

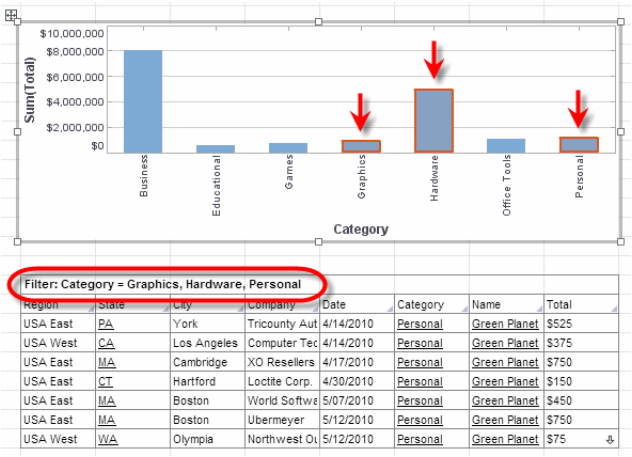
```
var conds = Query1.dataConditions;
var txt = 'Filter: None'

if(!isNull(conds)) {
  txt = 'Filter: ' + conds[0].attr + ' = ';
  for(var i=0;i<conds.length;i++) {
```

```
    txt += conds[i].value;
    if(i != conds.length-1) {
      txt += ', '
    }
  }
}

Query1.title = txt;
```

When the user selects groups on the Chart, the Table's title bar indicates the current flyover filter applied to the Table.



See Also

Flyover View, for additional information on flyover views.

ENABLED

Specifies whether the user can interact with the component. A disabled component appears dimmed when the user opens the Viewsheet.

Type

Boolean `true = enable, false = disable`

Example

```
enabled = false;
```

See Also

General Tab, for additional information on disabling a component.

FONT

Specifies the font for this element.

Type

`java.awt.Font`

Example

```
font = java.awt.Font('Verdana',java.awt.Font.BOLD, 35)
```

See Also

Font Tab, for more information about setting font.

FOREGROUND

Specifies the foreground (text) color of the component.

Type

`java.awt.Color`	`e.g., java.awt.Color.BLUE`
`number (hex)`	`e.g., 0xFF0000`
`string (color name)`	`e.g., 'red'`
`array [r,g,b]`	`e.g., [255,0,0]`
`JSON {r:#,g:#,b:#}`	`e.g., {r:255,g:0,b:0}`

Example

```
foreground = 'red';
```

See Also

Color Tab, for more information about setting color.

FORMAT

Sets the format. The available formats are shown below:

- `StyleConstant.DATE_FORMAT` for date display
- `StyleConstant.DECIMAL_FORMAT` for number display
- `StyleConstant.MESSAGE_FORMAT` for text display
- `StyleConstant.PERCENT_FORMAT` for fraction display
- `StyleConstant.CURRENCY_FORMAT` for locale-adapting currency display

To format cells or columns of a Table, use the *tablelens* property.

For the Date, Decimal, and Message formats, use the *`formatSpec`* property to assign a format specification using the appropriate masks.

Example (Number Format)

```
format = StyleConstant.DECIMAL_FORMAT;
formatSpec = "#,###.00";
```

Example (Text Format)

```
format = StyleConstant.MESSAGE_FORMAT;
formatSpec = "{0} Inc."
```

FORMATSPEC

Specifies a formatting instruction to be used for representing data on the screen. The formatting instruction is specific to the format type, Date, Number, or Text.

To format cells or columns of a Table, use the *tablelens* property.

Example (Date Format)

```
format = StyleConstant.DATE_FORMAT;
formatSpec = "MM/dd/yyyy";
```

Example (Number Format)

```
format = StyleConstant.DECIMAL_FORMAT;
formatSpec = "#,###.00";
```

Example (Text Format)

```
format = StyleConstant.MESSAGE_FORMAT;
formatSpec = "{0} Inc."
```

Date masks follow the *java.text.SimpleDateFormat* format.

Date Format: For date formatting, use the following date masks:

```
M = Month
d = date
y = year
E = day of the week
```

Example:

```
For the date Nov 8, 2006:

M = 11; MM = 11; MMM = Nov; MMMM = November
d = 8; dd = 08;
yy = 06; yyyy = 2006
EEE = Wed
EEEE = Wednesday

Therefore, mask 'MMM-dd-yyyy' yields “Nov-08-2006.”
```

Numeric masks follow the *java.text.DecimalFormat* format.

Number Format: For number formatting, use the following numeric masks:

```
# = number
0 = Number with zero padding
```

Example:

```
For the number 124521.63:

#,###.##  yields 124,521.63
#,###.000 yields 124,521.630
```

Text Format: For text formatting, use `{0}` as a placeholder for the string data.

Example:

```
For the string 'Susan':

Salesperson: {0}  yields Salesperson: Susan
--{0}-- yields --Susan--
```

See Also

formatDate, for details on date formatting syntax.
formatNumber, for details on decimal formatting syntax.
Format Tab, in *Advanced Dashboard Design*, for more information about format specifications.

POSITION

Specifies the position of the top-left corner of a component in terms of Viewsheet grid cells, where the top-left corner of the grid is (0,0). Fractional values for vertical and horizontal position are rounded down to the next lowest integer. This property does not take effect in the design view of Visual Composer and does not apply to components within a *Selection Container*.

Type

value	a `java.awt.Point` object: `java.awt.Point(horizontal,vertical)`

Example

```
// Table with top-left corner at column 4, row 9
TableView1.position = java.awt.Point(4,9)

// Chart with top-left corner at column 2, row 25
Chart1.position = java.awt.Point(2,25)
```

SELECTEDLABEL

Specifies the *selected label* of the component (not the value). Applies to single value input elements (Slider, Spinner, Radio Button, Combo Box) and the tabs of co-located components (Tabbed Interface). This property is readable and writable.

Type

`String`	`label of the selected option`

Example

```
RadioButton1.selectedLabel = 'New Jersey';
```

See Also

lables, to access the labels displayed in the component.
selectedObject, to access the value corresponding to the label.
selectedLabels, to access the currently selected labels in multi-valued components.
Accessing User Input in Script, for more information about `selectedLabel`.

SELECTEDLABELS

Returns the *selected labels* (not the values) of the component as an array. Applies to multi-value input elements (Check Boxes) and selection elements (Selection Lists, Selection Trees, Range Slider, Calendars). This is a read-only property.

Type

`Array`	`array of selected option labels`

Example

```
var num = SelectionList1.selectedLabels.length;
var msg = 'You selected ' + num + ' objects: ';
msg += SelectionList1.selectedLabels.join(', ');
alert(msg)
```

See Also

lables, to access the labels displayed in the component.
selectedObjects, to access the values corresponding to the labels.
selectedLabel, to access the currently selected label in single-valued components.
Accessing User Input in Script, for more information about `selectedLabels`.

SELECTEDOBJECT

Specifies the *selected value* of the component (not the label). Applies to single value input elements (Slider, Spinner, Radio Button, Combo Box) and the tabs of co-located components (Tabbed Interface). This property is readable and writable.

Type

```
String              value of the selected option
```

Example

```
RadioButton1.selectedObject = 'NJ';
```

See Also

values, to access the values presented in the component list.
Accessing the Most Recent User Modification, to retrieve the most recent modified object.
selectedLabel, to access the label corresponding to the value.
selectedObjects, to access the current selection in multi-valued components.
Accessing User Input in Script, for more information about `selectedObject`.

SELECTEDOBJECTS

Specifies the *selected values* (not the labels) of the component as an array. Applies to multi-value input elements (Check Boxes) and selection elements (Selection Lists, Selection Trees, Range Slider, Calendars).

This property is readable and writable for Selection Lists and Selection Trees, and is read-only for other components. Script that reads the value of the `selectedObjects` property can be placed either at the element level or in the *onRefresh Handler*, while script that writes the `selectedObjects` property must be placed in the *onRefresh Handler*.

Type

Array	array of selected option values

Example

```
var num = SelectionList1.selectedObjects.length;
var msg = 'You selected ' + num + ' objects: ';
msg += SelectionList1.selectedObjects.join(', ');
alert(msg)
```

See Also

values, to access the values presented in the component list.

Accessing the Most Recent User Modification, to retrieve the most recent modified object.

selectedLabels, to access the labels corresponding to the values.

selectedObject, to access the current selection in single-valued components.

Accessing User Input in Script, for more information about `selectedObjects`.

SIZE

Specifies the width and height of a component in terms of Viewsheet grid cells. Fractional values for width and height are rounded down to the next lowest integer. This property does not take effect in the design view of Visual Composer and does not apply to components within a *Selection Container*.

Type

value	a `java.awt.Dimension` object: `java.awt.Dimension(columns,rows)`

Example

```
// Table spanning 7 grid columns, 4 grid rows
TableView1.size = java.awt.Dimension(7,4)

// Chart spanning 6 grid columns, 20 grid rows
Chart1.size = java.awt.Dimension(6,20)
```

TITLE

Specifies the text that appears in the title bar of the component. This property is only available for components that possess a title bar.

Type

String	text to appear in title bar

Example

```
SelectionList1.title = "Company Selector"
```

See Also

General Tab, in *Advanced Dashboard Design*, for more information on the title.

VISIBLE

Specifies the visibility of the component. This property can be used to dynamically hide elements in a Viewsheet.

Type

Boolean	true: display the component false: hide the component

Example

```
if (Gauge1.value > 200) {
  Text1.text = 'Too high!';
  Text1.visible = true;
}
else if (Gauge1.value < 50) {
  Text1.text = 'Too low!';
  Text1.visible = true;
}
else {
  Text1.visible = false;
}
```

See Also

General Tab, in *Advanced Dashboard Design*, for more information on visibility.

WRAPPING

Specifies whether text that exceeds its boundaries is wrapped or truncated. This property is available for the following components: Table, Embedded Table, Crosstab Table, Selection List, Selection Tree, and Text.

Type

Boolean	true: wrap text false: truncate text

Example

```
Table1.wrapping = true;
```

See Also

General Tab, in *Advanced Dashboard Design*, for more information on visibility.

VS.3 Table

This section presents functions for Table operations.

See Also

Crosstab Table, for properties specific to pivot tables.

COMMIT(TYPE)

When a user adds or changes a row in an editable table, the new or modified row is highlighted in color. To clear the highlight on these rows

and remove them from the corresponding array returned by `getFormRows`, use the table's '`commit`' function.

Parameters

type the type of rows to commit: ADDED, CHANGED, DELETED, or the index of a particular row

Example

```
// Clear highlights on added rows, and remove them from
// array returned by getFormRows(ADDED).
Table1.commit(ADDED);
// Clear highlights on changed rows, and remove them from
// array returned by getFormRows(CHANGED).
Table1.commit(CHANGED);

// Removes deleted rows from array returned by
// getFormRows(DELETED).
Table1.commit(DELETED);

// Removes the row returned by getFormRows(idx) from the
// array of modified rows.
Table1.commit(idx);
```

See Also

Accessing User-Modified Data in a Table, for information on accessing user edits.

Table Properties: Advanced Tab, for information on making a table editable.

DATA

Two dimensional array containing the original table data (prior to grouping and aggregation) If no grouping or summarization is defined for the table, the '`data`' property is the same as the '*`table`*' property. Otherwise, '`data`' holds the raw data and '`table`' holds the summarized data. The data can be referenced using the row number as the first array index and the column number or label as the second array index. This property is read-only.

Type

`Two-dimensional array`

Example 1

```
for (var i=1; i<data.length; i++) {
  if (Query1.data[i]['Name'] == 'Web Bridge') {
    alert('Note: "Web Bridge" will be discontinued.');
  }
}
```

Example 2

```
Text1.text = 'Grand total is ' + sum(Query1.data['Total']);
```

See Also

Accessing Table Data, for information about working with table data.

DATA.LENGTH

Returns the number of rows in a table, prior to aggregation and summarization.

Type

integer

Example

```
// Testing for empty result set:
if (Query1.table.length == 1) {
  Text1.text = 'No data returned';
}
```

See Also

table.length, for the length of post-aggregate (as-displayed) results table.
Accessing Table Data, for information about working with table data.

DATA.SIZE

Returns the number of columns in a table, prior to grouping and aggregation.

Type

integer

See Also

table.size, for the width of post-aggregate (as-displayed) results table.
Accessing Table Data, for information about working with table data.

GETFORMROW(ROW)

For an editable table, retrieves the edited row specified by the index. Often the index is provided by the *event* object for the editable table.

Parameters

row row index

Example

```
var changedRow = Table1.getFormRow(row);
```

See Also

Accessing the Most Recent User Modification, for an example of using getFormRow().
Accessing User-Modified Data in a Table, for general discussion.
Table Properties: Advanced Tab, for information on making a table editable.

GETFORMROWS([TYPE])

For an editable table, retrieves the rows that have been modified by the user. The `type` parameter specifies the set of rows to return, and can be provided in string form (e.g., `'added'`) or constant form (e.g., `ADDED`). Omit the parameter to return all rows in table.

Parameters

type	ADDED: return new rows inserted by user CHANGED: return old rows modified by user DELETED: return old rows deleted by user OLD: return all old rows

Example

```
var allRows = Table1.getFormRows();
var addedRows = Table1.getFormRows(ADDED);
var changedRows = Table1.getFormRows(CHANGED);
var deletedRows = Table1.getFormRows(DELETED);
var oldRows = Table1.getFormRows(OLD);
```

See Also

Accessing User-Modified Data in a Table, for general discussion.
Table Properties: Advanced Tab, for information on making a table editable.

APPENDROW(ROW)

For an editable table, inserts a new blank row following the row specified by integer parameter `row`, and adds this new row to the array returned by `getFormRows(ADDED)`.

Parameters

row	integer row index

Example

```
// Add new row below row 4:
TableView1.appendRow(4);

// Set initial values in new row (row 5):
TableView1.setObject(5,0,'Enter Name')
TableView1.setObject(5,1,'Enter Price')
TableView1.setObject(5,2,'Enter Quantity')
TableView1.setObject(5,3,'Enter Total')
```

See Also

setObject(row,col,value), to write data into an editable table cell.
Accessing User-Modified Data in a Table, for general discussion.
Table Properties: Advanced Tab, for information on making a table editable.

DELETEROW(ROW)

For an editable table, deletes the row specified by integer parameter `row`, and adds this deleted row to the array returned by `getFormRows(DELETED)`.

Parameters

row integer row index

Example

```
// Delete row 4:
TableView1.deleteRow(4);
```

See Also

Accessing User-Modified Data in a Table, for general discussion.
Table Properties: Advanced Tab, for information on making a table editable.

INSERTROW(ROW)

For an editable table, inserts a new blank row prior to the row specified by integer parameter `row`, and adds this new row to the array returned by `getFormRows(ADDED)`.

Parameters

row integer row index

Example

```
// Add new row above row 4:
TableView1.insertRow(4);

// Set initial values in new row 4:
TableView1.setObject(4,0,'Enter Name')
TableView1.setObject(4,1,'Enter Price')
TableView1.setObject(4,2,'Enter Quantity')
TableView1.setObject(4,3,'Enter Total')
```

See Also

setObject(row,col,value), to write data into an editable table cell.
Accessing User-Modified Data in a Table, for general discussion.
Table Properties: Advanced Tab, for information on making a table editable.

GETHIDDENCOLUMNVALUE(COL,ROW)

For a hidden column (often a primary key column) in an editable table, retrieves the value in the specified row of the hidden column.

Parameters

col hidden column name
row row index

Example

```
for(var i=0; i<TableView1.table.length; i++) {
  if(TableView1.getFormRow(i).rowState == CHANGED) {
    var hiddenValue =
TableView1.getHiddenColumnValue("columnName", i);
  }
}
```

See Also

Hiding Table Columns, in *Advanced Dashboard Design*, to hide columns of an editable table.

Accessing the Most Recent User Modification, for an example of using getFormRow().

Table Properties: Advanced Tab, for information on making a table editable.

SETACTIONVISIBLE(NAME, BOOLEAN)

Sets the visibility/accessibility of end-user Table features. To disable a feature, set the value to `false`.

Parameters

name	name of table function (String): 'Export' 'Show Details' 'Sort Column' (column header 'Sort' button)
boolean	false: hide the specified function true: show the specified function

For a Crosstab, the `'Sort Column'` setting overrides the **`sort.crosstab.aggregate`** and `sort.crosstab.dimension` property settings.

Example

```
Table1.setActionVisible("Show Details", false);
```

See Also

thisViewsheet.setActionVisible(name,Boolean), to show or hide Viewsheet toolbar buttons.

ISACTIONVISIBLE(NAME)

Returns the visibility/accessibility of the specified end-user Table feature.

Parameters

name	name of table function (String): 'Export', 'Show Details'

Example

```
if (TableView1.isActionVisible('Export')==true) {
  Text1.text = 'Press toolbar button to export.';
}
```

See Also

thisViewsheet.isActionVisible(name), to return Viewsheet toolbar button visibility.

SETHYPERLINK(ROW, COL, HYPERLINK)

Creates a hyperlink for an individual table cell.

Parameter

row	row index
col	column index
hyperlink	string or inetsoft.report.Hyperlink.Ref object

Example

```
setHyperlink(1, 1, "Tutorial/Ad Hoc");
setHyperlink(1, 1, null); // remove hyperlink
```

See Also

Hyperlinks, in *Advanced Dashboard Design*, for basic information on hyperlinks.

SETCOLUMNWIDTH(COL, WIDTH)

Sets the width of the specified table column in pixels.

Parameter

col	column index (first column is index 0)
width	positive integer sets column width in pixels -1 sets automatic width calculation (default) 0 sets zero width (column hidden)

Example

```
//Set third column width to 200 pixels
TableView1.setColumnWidth(2,100);
```

SETROWHEIGHT(ROW, HEIGHT)

Sets the height of the specified table row in pixels. Row index 0 is the column header row. Setting the height for any body row has the effect of setting the same height for all body rows.

Parameter

row	row index (column header row is index 0)
height	positive integer sets row height in pixels -1 sets automatic height calculation (default) 0 sets zero height (row hidden)

Example

```
//Set body row height to 30 pixels
TableView1.setRowHeight(1,30);
```

SETOBJECT(ROW,COL,VALUE)

Sets the contents of an editable table cell to the specified value.

Parameter

row	row index of cell to modify
col	column index of cell to modify
value	value to be placed into cell

Example

```
// Insert text “Robert” into third row, fourth column:
TableView1.setObject(2,3,'Robert');
```

See Also

Table Properties: Advanced Tab, in *Advanced Dashboard Design*, for information on how to make a table editable.

Accessing the Most Recent User Modification, to retrieve the most recent modified object.

Accessing User Input in Script, for information about `selectedObject`.

SETPRESENTER(ROW,COL,PRESENTER)

Sets the contents of an table cell to the specified presenter. Built-in presenters include the following:

– inetsoft.report.painter.BarPresenter
– inetsoft.report.painter.IconCounterPresenter
– inetsoft.report.painter.BooleanPresenter
– inetsoft.report.painter.Bar2Presenter
– inetsoft.report.painter.ButtonPresenter
– inetsoft.report.painter.ShadowPresenter

Parameter

row	row index of cell to modify
col	column index of cell to modify
presenter	name of Presenter

Example

```
// Rotate the text in the top-left cell by 45 degrees:
Table1.setPresenter(0,0,"HeaderPresenter");
```

TABLE

Two dimensional array holding the table cell data *as displayed* (after any grouping and aggregation). If no grouping or summarization is defined on the table, the ‘`table`’ property is the same as the ‘*`data`*’ property. The data can be referenced using the row number as the first array index and the column number or name as the second array index.

This property is read-only. To write data to an editable table, use *setObject(row,col,value)*.

Type

```
Two-dimensional array
```

Example 1

```
for (var i=1; i<table.length; i++) {
  if (Query1.table[i]['Name'] == 'Web Bridge') {
    alert('Note: "Web Bridge" will be discontinued.');
  }
}
```

Example 2

```
Text1.text = 'Grand total is ' + sum(Query1.table['Total']);
```

See Also

setObject(row,col,value), for information on modifying data in an editable table.
Accessing Table Data, for information about working with table data.

TABLE.LENGTH

Returns the number of rows in a table, as displayed. It counts the header rows as well as the data rows.

Type

```
integer
```

Example

```
// Testing for empty result set:
if (Query1.table.length == 1) {
  Text1.text = 'No data returned';
}
```

See Also

Accessing Table Data, for information about working with table data.
data.length, for number of rows prior to grouping and summarization.

TABLE.SIZE

Returns the number of columns in a table, as displayed.

Type

```
integer
```

See Also

Accessing Table Data, for information about working with table data.
data.size, for number of columns prior to grouping and summarization.

TABLESTYLE

Specifies a predefined style to be applied to the Table.

Type

```
String
```

Example

```
Query1.tableStyle = 'Shaded'
```

See Also

Table Properties, for more information about Table propeties.

TABLELENS

Specifies the Table's TableLens object, which allows you to access and modify table attributes using API methods.

Type

```
inetsoft.report.lens.AttributeTableLens
```

Example

```
TableView1.tablelens.setForeground(0, 2,
java.awt.Color.red);
```

SHRINK

Reduces the height of the table to match the number of records displayed. For example, if the original table height (as designed) was 10 rows, and the user then makes selections which restrict the result set to five rows, the table's bottom border will retract upward or "shrink" to eliminate any empty rows at the bottom of the table.

Type

```
Boolean              true: shrink table to fit contents
                     false: do not shrink table
```

Example

```
TableView1.shrink=true;
```

See Also

Table Properties: Advanced Tab, in *Advanced Dashboard Design*, for information on setting the property without script.

VALUE

Returns the value in the cell currently referenced by a 'Format' expression script. For example, if you specify a 'Text Color' expression in the 'Format' dialog box for a table header cell, you can use the `value` property within the script to obtain the value in the targeted header cell. Likewise, if you specify a 'Text Color' expression in the 'Format' dialog box for a detail cell in a table column (which means that the expression will be evaluated in turn for each detail cell in the column), you can use the `value` property within the script to obtain the value of the currently referenced detail cell.

Example

The script below can be used in a 'Text Color' or 'Fill Color' expression to set color formatting on a table detail cell.

```
if (value > 1000) {
  [255,0,0]
}
else {
  [0,255,0]
}
```

See Also

Accessing Data in the Current Cell ('value'), for further information.
Table Properties, for general information about the Table component.

FIELD

Returns the value of the cell in the specified field (column) and same row as the cell currently referenced by a 'Format' expression script. For example, if you specify a 'Text Color' expression in the 'Format' dialog box for a detail cell in a table column, you can use the `field` property within the script to obtain the value of the detail cell in the corresponding row of another column.

Example

The script below can be used in a 'Text Color' or 'Fill Color' expression to set color formatting on a table detail cell.

```
if (field['Total'] > 400) {
  [255,0,0];
} else {
  [0,0,255];
}
```

See Also

Accessing Data in the Same Row ('field'), for further information.
Table Properties, for general information about the Table component.

COL

Returns the column index of the cell currently referenced by a 'Format' expression script. For example, if you specify a 'Text Color' expression in the 'Format' dialog box for a detail cell in a table, you can use the `col` property within the script to obtain the index of the column. This, together with *table* array and the *row* property allow you to construct relative references to other cells in the table.

Example

The script below can be used in a 'Text Color' or 'Fill Color' expression to set color formatting on a table detail cell. It assigns a red color if the value

in the cell is less than the value of the same cell in the row above, and assigns a green color otherwise.

```
if (row > 1) {
  if (value < table[row-1][col]) {
    [255,0,0]
  }
  else {
    [0,255,0]
  }
}
```

See Also

table, for more information about the 'table' array.

Accessing Data in Different Row or Column ('row'/'col'), for further information.

Table Properties, for general information about the Table component.

ROW

Returns the row index of the cell currently referenced by a 'Format' expression script. For example, if you specify a 'Text Color' expression in the 'Format' dialog box for a detail cell in a table, you can use the `row` property within the script to obtain the index of the row. (The index will change as the expression is evaluated in turn for each detail cell in the column.) This, together with *table* array and the *col* property allow you to construct relative references to other cells in the table.

Example

The script below can be used in a 'Text Color' or 'Fill Color' expression to set color formatting on a table detail cell. It assigns a red color if the value in the cell is less than the value of the same cell in the row above, and assigns a green color otherwise.

```
if (row > 1) {
  if (value < table[row-1][col]) {
    [255,0,0]
  }
  else {
    [0,255,0]
  }
}
```

See Also

table, for more information about the 'table' array.

Accessing Data in Different Row or Column ('row'/'col'), for further information.

Table Properties, for general information about the Table component.

CELLFORMAT

Specifies the Table cell data format as a two dimensional array. The attribute can be referenced using the row number as the first index and the column header or number as the second index, `[row][col]`.

Type

```
two dimensional array of formats
```

Example

Column and row indices start at 0.

```
cellFormat[5][3] = [Chart.DATE_FORMAT, "yy-MMMM-dd"];
cellFormat[2]['Total'] = [Chart.DECIMAL_FORMAT,
"#,###.00"];
```

See Also

Table Properties, for general information about the Table component.

COLFORMAT

Specifies the Table column data format as an array. The attribute can be referenced using the column header or number.

Type

```
array of formats
```

Example

Column indices start at 0.

```
colFormat[5] = [Chart.DATE_FORMAT, "yy-MMMM-dd"];
colFormat['Total'] = [Chart.DECIMAL_FORMAT, "#,###.00"];
```

See Also

Table Properties, for general information about the Table component.

TIPVIEW

Sets the data tip for the Table. The data tip is a graphical form of tooltip which uses an arbitrary Viewsheet component to adaptively display summarized data when the user hovers the mouse over cells in the Table. See *Data Tip View* in *Advanced Dashboard Design* for more details.

Type

```
String        name of a Viewsheet component
```

Example

```
TableView1.tipView='Gauge1';
```

See Also

Data Tip View, in *Advanced Dashboard Design*, to set the property without using script.

FLYOVERVIEWS(ARR)

Specifies an array of component names, representing the components to be filtered by table flyover.

Parameters

arr	array of component names (String)

Example

```
TableView1.flyoverViews = ['Crosstab1','Gauge1'];
```

See Also

Flyover View, in *Advanced Dashboard Design*, to set the property without using script.

FLYONCLICK

Specifies that components filtered by a table flyover will be updated only when the user clicks on the table.

Type

Boolean	true: update on mouse click false: update on mouse hover (default)

Example

```
TableView1.flyOnClick = true;
```

See Also

Flyover View, in *Advanced Dashboard Design*, to set the property without using script.

VS.4 Crosstab Table

This section presents functions specifically for Crosstab Table operations. The Crosstab Table component also supports all of the properties of the *Table* component.

See Also

Crosstabs, for general information about crosstab or pivot tables.

DRILLENABLED

Specifies whether the automatic drill-down feature is enabled for the crosstab. If disabled, the '+' and '–' drill operators do not appear on the Crosstab headers.

Type

Boolean	true = enabled, false = disabled

Example

```
Crosstab1.drillEnabled = false;
```

See Also

Drilling Down into a Crosstab, in *Advanced Dashboard Design*, for information on using the drill-down feature.

FILLBLANKWITHZERO

Populate empty cells of the crosstab table with zeros. (Empty data cells occur when no results exist corresponding to the row and column headings.)

Type

Boolean	true: use zeros in empty cells false: leave empty cells blank

Example

```
fillBlankWithZero = true
```

SUMMARYSIDEBYSIDE

Display multiple measures side-by-side rather than stacked vertically (the default).

Type

Boolean	true: place measures side-by-side false: stack measures vertically (default)

Example

```
summarySideBySide = true
```

VS.5 Chart

This section presents Viewsheet-specific properties for Chart components. See *Chart Script Reference* for details about properties available in both Viewsheet and report Chart scripts.

See Also

Chart Script Tutorial, for examples of complex chart scripting.

DRILLENABLED

Specifies whether the automatic drill-down feature is enabled for the chart. If disabled, the '+' and '–' drill operators do not appear on the chart axes.

Type

Boolean	true = enabled, false = disabled

Example

```
Chart1.drillEnabled = false;
```

See Also

Drilling Down into a Chart, in *Advanced Dashboard Design*, for information on using the drill-down feature.

FLYOVERVIEWS(ARR)

Specifies an array of component names, representing the components to be filtered by chart flyover.

Parameters

arr	array of component names (String)

Example

```
Chart1.flyoverViews = ['TableView1','Gauge1'];
```

See Also

Flyover View, in *Advanced Dashboard Design*, to set flyover in the 'Chart Properties' dialog box.

FLYONCLICK

Specifies that components filtered by a chart flyover will be updated only when the user clicks on the chart.

Type

Boolean	true: update on mouse click false: update on mouse hover (default)

Example

```
Chart1.flyOnClick = true;
```

See Also

Flyover View, in *Advanced Dashboard Design*, to set flyover in the 'Chart Properties' dialog box.

TIPVIEW

Sets the data tip for the Chart. The data tip is a graphical form of tooltip which uses an arbitrary Viewsheet component to adaptively display summarized data when the user hovers the mouse over cells in the Chart. See *Data Tip View* in *Advanced Dashboard Design* for more details.

Type

String	name of a Viewsheet component

Example

```
Chart1.tipView='Gauge1';
```

See Also

Data Tip View, in *Advanced Dashboard Design*, to set the property without using script.

SETACTIONVISIBLE(NAME,BOOLEAN)

Sets the visibility/accessibility of end-user Chart features. To disable a feature, set the value to `false`.

Parameters

name	name of chart function (String): 'Brush', 'Edit', 'Maximize', 'Show Data', 'Show Details', 'Zoom'
boolean	false: hide the specified function true: show the specified function

Example

```
Chart1.setActionVisible("Show Details", false);
```

See Also

thisViewsheet.setActionVisible(name,Boolean), to show or hide Viewsheet toolbar buttons.

ISACTIONVISIBLE(NAME)

Returns the visibility/accessibility of the specified end-user Chart feature.

Parameters

name	name of chart function (String): 'Brush', 'Edit', 'Maximize', 'Show Data', 'Show Details', 'Zoom'

Example

```
if (Chart1.isActionVisible('Edit')==true) {
  Text1.text = 'Click the Edit button to open Chart
Editor';
}
```

See Also

thisViewsheet.isActionVisible(name), to return Viewsheet toolbar button visibility.

TOOLTIP

Specifies a tooltip to be displayed on chart elements. You can use placeholders `{0}`, `{1}`, `{2}`, etc. to insert data values from the corresponding fields into the tooltip text, and format the inserted data using java.text.MessageFormat format. You may additionally use HTML markup such as "`<i>`", "`<b>`," and "`<br>`" to format the tooltip contents.

Type

String	The tooltip to display

Example

```
Chart1.toolTip = "<b>Company</b>: {0} <br><b>Total Sales</
b>: {1,number,$##,###.00}";
```

DATA

Two dimensional array containing the chart data. This array is readable and writable.

Type

```
Two-dimensional array
```

Example

```
for (var i=1; i<data.length; i++) {
  if (Chart1.data[i]['Name'] == 'Web Bridge') {
    Chart1.data[i]['Sum(Quantity Purchased)'] = 0;
    alert('Note: "Web Bridge" has been discontinued.');
  }
}
```

To bind an unbound chart to a Data Block or cube in the Worksheet, use the following syntax:

```
// Bind a Data Block:
Chart1.data="table_name";

// Bind a cube:
Chart1.data="cube::table_name";
```

See Also

Data, in *Chart Script Reference*, for information about working with chart data.

VS.6 Selection List and Tree

This section presents properties for Selection List and Selection Tree components. The Selection Tree shares all of the properties of the Selection List with the exception of `singleSelection`.

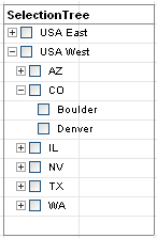
See Also

Selection List and *Selection Tree*, for more information about selection elements.

DRILLMEMBER

Does not apply to Selection Lists.

Returns the name of the field *one level below* the deepest level in the hierarchy at which the user has made a selection. For example, consider a three-level Selection Tree hierarchy: ‘Region’ > ‘State’ > ‘City’.



If the user's most specific selection is at the level of *region*, then `drillMember` returns `'State'`. If the user's most specific selection is at the level of *state*, then `drillMember` returns `'City'`. If the user's most specific selection is at the level of *city*, then because there is no lower level `drillMember` again returns `'City'`. If the user has not selected any values in the Selection Tree, then `drillMember` returns the highest level, here `'Region'`.

Type

`String`	`field name for child of deepest selected level`

Example

```
var drillField = SelectionTree1.drillMember;
```

This property is useful for creating interactive chart drilldowns.

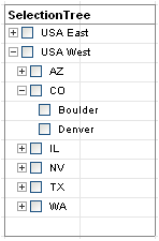
See Also

Walkthrough: Interactive Chart Drilldown, for an example usage.
Selection List Properties and *Selection Tree Properties*, for more information.

DRILLMEMBERS

Does not apply to Selection Lists.

Returns an array of the levels at which the user has made a selection. The array includes all levels from the top level to one level below the current selection. (The top level is included even when no selection is made.) For example, consider a three-level Selection Tree hierarchy: 'Region' > 'State' > 'City'.



If the user makes a selection at the level of *region*, `drillMembers` returns the array `['Region','State']`. If the user makes a selection at the level of *state* or *city*, `drillMembers` returns the array `['Region','State','City']`. This array is useful for displaying multiple geographical levels on a map, as shown in the example below.

Type

`Array`	`array of field names for selected levels`

Example

Follow the steps below to create a dynamic multi-level map drill:

1. Create a new Viewsheet based on the ‘Sales’ > ‘Sales Explore’ Worksheet.

2. From the **Component** panel, drag a Chart component into the Viewsheet grid. Resize the Chart as desired.

3. From the **Component** panel, drag a Selection Tree component into the Viewsheet grid.

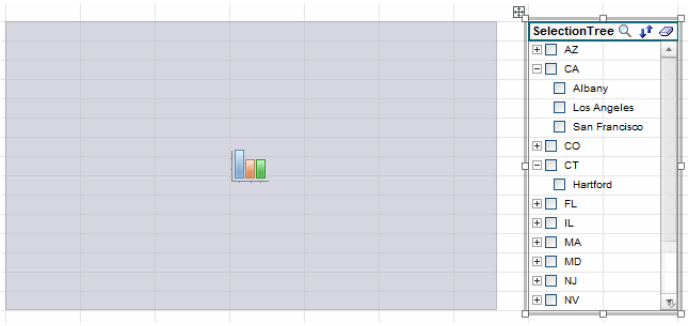
4. Add a two-level geographical hierarchy to the Selection Tree. Follow the steps below:

   a. Expand the ‘Sales’ data block in the **Component** panel.

   b. Drag the ‘State’ field from the data block onto the Selection Tree. This adds the ‘State’ values to the tree.

   c. Drag the ‘City’ field from the data block onto the Selection Tree. This adds the ‘City’ values to the tree.

   d. Resize the Selection Tree so that it can display a reasonable number of options.

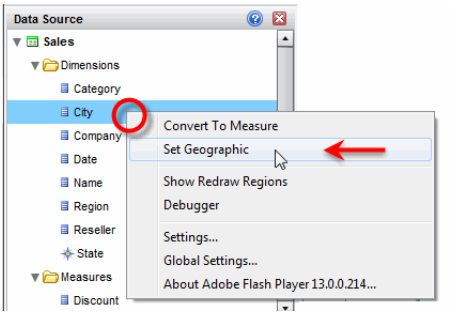


5. Press the ‘Edit’ button on the Chart to open the Chart Editor.

6. Press the ‘Select Chart Style’ button in the **Data** panel, and choose the ‘Map’ chart type.

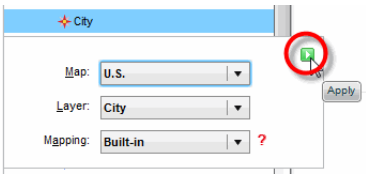
7. In the **Data Source** panel, right click on the ‘State’ field, and select ‘Set Geographic’.

8. In the **Data Source** panel, right click on the ‘City’ field, and select ‘Set Geographic’.



The red question mark indicates that not all cities in the 'City' field of this example have mappings.

9. In the pop-up panel, press the green 'Apply' button.

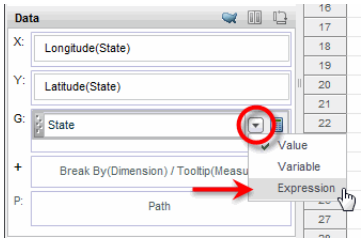


10. From the **Data Source** panel of the Chart Editor, drag the 'Total' measure to the 'Color' region in the **Visual** panel.

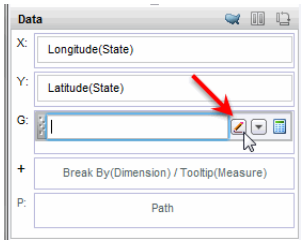
11. From the **Data Source** panel of the Chart Editor, drag the 'State' dimension to the 'Geographic' region in the **Data** panel.

12. Configure the chart to adapt the geographical level based on the user's selection in the Selection Tree. To do this, follow the steps below:

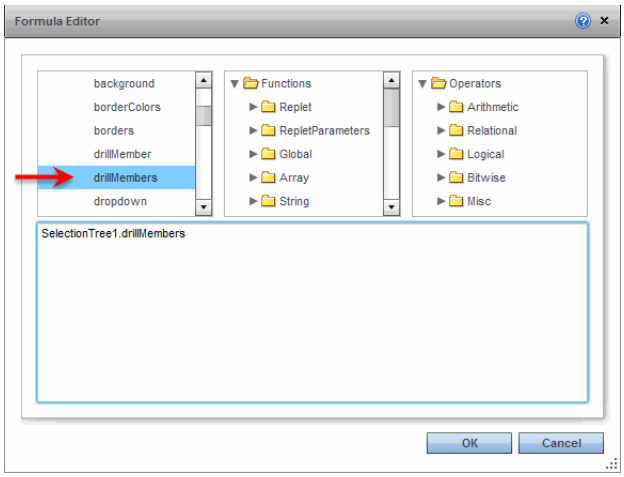
    a. Press the "arrow" button next to the 'State' dimension in the **Data** panel. In the pop-up menu, select the 'Expression' option.



    b. Press the 'Edit' button to open the Script Editor.

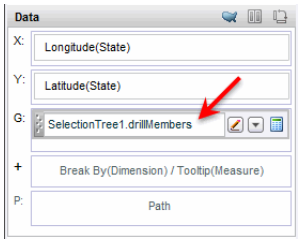


c. In the top-left panel of the Script Editor, expand the 'SelectionTree1' node, and click 'drillMembers' (with a final 's'). This adds the SelectionTree1.drillMembers property in the editing area.



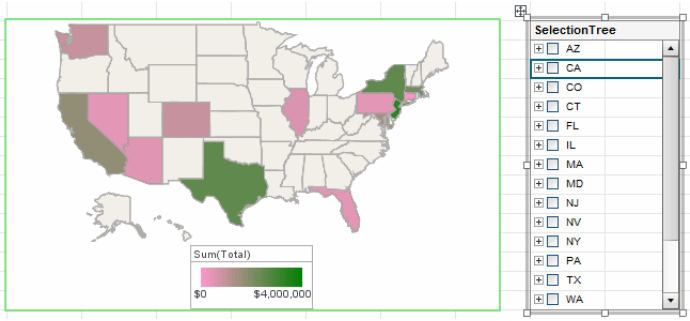
If the user makes no selection in the Selection Tree, the drillMembers property returns the array ['State'], which displays the 'Total' measure on a state-basis. If the user selects one or more states or cities in the Selection Tree, drillMembers returns the array ['State','City'], which displays the 'Total' measure on a city-basis.

d. Press 'OK' to close the Script Editor. The output of SelectionTree1.drillMembers is now assigned to the chart Geographical binding.

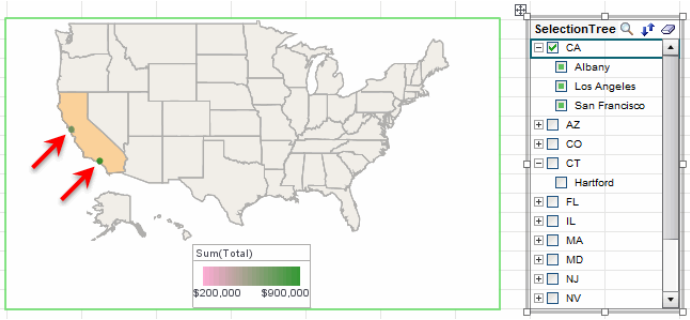


13. Verify that the map adapts to the user's selections:

a. Press the 'Clear Selection' button in the Selection Tree to remove any existing selections. Observe that the map represents the measure on a state level, the top level of the Selection Tree hierarchy.



b. Select the 'CA' option in the Selection Tree. Observe that the Selection Tree filters the map to display only data from California. In addition, `drillMembers` property changes the geographical binding to represent the 'City' dimension, and the measure is now displayed at the city-level.



To automatically zoom a map to display only the regions that the user selects, see the example for *Coordinate.setFullMap(boolean)*.

See Also

drillMember, to create a dynamic drilldown for a single geographical level.
Walkthrough: Interactive Chart Drilldown, for an example usage.
Selection List Properties and *Selection Tree Properties*, for more information.

DROPDOWN

Sets the Selection List to *dropdown* style, in which the Selection List is compactly represented by just its title bar. When the user moves the mouse over the List, The List automatically expands to display the available options.

Type

Boolean `true: show compact title-bar only display`
`false: show normal List display (default)`

Example

```
SelectionList1.dropdown = true;
```

See Also

Selection List Properties and *Selection Tree Properties*, for more information.

SETACTIONVISIBLE(NAME,BOOLEAN)

Sets the visibility/accessibility of end-user Selection features. To disable a feature, set the value to `false`.

Parameters

name	name of toolbar button (String): 'Apply', 'Clear Selection', 'Reverse Selection', 'Sort Selection',
boolean	false: hide the specified function true: show the specified function

Example

```
Selection1.setActionVisible("Reverse Selection", false);
```

See Also

thisViewsheet.setActionVisible(name,Boolean), to show or hide Viewsheet toolbar buttons.

SINGLESELECTION

Does not apply to Selection Trees.

Specifies that only a single choice can be selected in the list.

Type

Boolean	true: select only one item false: select one or more items (default)

Example

```
SelectionList1.singleSelection = true
```

See Also

Selection List Properties and *Selection Tree Properties*, for more information.

SORTTYPE

Specifies the sort order for the Selection List or Tree. You can list choices in ascending (alphabetical) order, in descending (reverse alphabetical) order, or display only compatible choices. This last option (the default) hides incompatible choices, but allows a user to display them if desired.

Type

Integer	order of items in Selection List: StyleConstant.SORT_ASC: Ascending order StyleConstant.SORT_DESC: Descending order StyleConstant.SORT_HIDE_OTHERS: Hide incompatible choices

Example

```
SelectionList1.sortType = StyleConstant.SORT_ASC;
```

See Also

Selection List Properties and *Selection Tree Properties*, for more information.

SUBMITONCHANGE

Specifies that user selections should be submitted immediately to the server and the Viewsheet refreshed with updated data (default). When this feature is disabled, user selections are not submitted until the user explicitly click the green 'Apply' button.

Type

Boolean	`true: submit user selections immediately` `false: submit user selection on 'Apply'`

Example

```
SelectionList1.submitOnChange = false;
```

See Also

Selection List Properties and *Selection Tree Properties*, for more information.

VS.7 Selection Container

This section discusses the properties that are unique to the Selection Container.

See Also

Common Properties, for properties shared with other components.
Selection Container, for more information about Selection Containers.

ADHOCENABLED

Determines whether end users can add new selection fields (from the Chart Editor) to the Selection Container.

Type

Boolean	`true: allow users to add new selections` `false: prevent users from adding selections`

Example

```
CurrentSelection1.adhocEnabled = false;
```

See Also

Selection Container Properties, for more information on user selection editing.

SHOWCURRENTSELECTION

Specifies whether the Selection Container displays the current state of all Viewsheet selection components.

Type

Boolean	`true: display all current selections` `false: do not display current selections`

Example

```
CurrentSelection1.showCurrentSelection = false;
```

See Also

Selection Container Properties, for more information on user selection editing.

VS.8 Range Slider

This section discusses the properties that are unique to the Range Slider.

See Also

Common Properties, for properties shared with other components.
Range Slider, for more information about Range Sliders.

CURRENTVISIBLE

Specifies that the value currently selected on the Range Slider is numerically displayed at all times. Otherwise, the current value is only numerically displayed while the user is actively dragging the slider.

Type

Boolean	`true: display currently selected value` `false: hide currently selected value`

Example

```
RangeSlider1.currentVisible = false;
```

See Also

Range Slider Properties, for more information on user selection editing.

LENGTH

Specifies the initial slider range. For a numerical field, this gives the range as a multiple of the minimum length *rangeSize*. For example, if `length` is 3 and `rangeSize` is 100, the initial slider range is 300. For a date field, the units are specified by the *rangeType* setting (year, month, or day). In either case, the user can subsequently readjust the range as desired.

Type

Number	`Slider size, as multiple of minimum length`

Example

```
RangeSlider1.length = 3;
```

See Also

Range Slider Properties, for more information on user selection editing.

RANGESIZE

Specifies the minimum slider range for a numerical field, the smallest interval that the user is able to select when interactively adjusting the range.

Type

Number Minimum slider range

Example

```
RangeSlider1.rangeSize = 1000;
```

See Also

Range Slider Properties, for more information on user selection editing.

MAXRANGESIZE

Specifies the maximum slider range, the largest interval that the user is able to select when interactively adjusting the range. The units are the same as those of the *length* property.

Type

Number Maximum slider range

Example

```
RangeSlider1.maxRangeSize = 10;
```

See Also

Range Slider Properties, for more information on user selection editing.

MAX

Specifies the maximum slider value. If 'Single Value' is selected under the **Data** tab of the 'Range Slider Properties' dialog box, the `max` value is a single number or date (as appropriate for the data represented on the Slider). If 'Composite Value' is selected under the **Data** tab, the `max` value is an array.

Type

Number, Date, or Array

Examples

```
RangeSlider1.max = 10; // number
RangeSlider1.max = new Date("December 31, 2009"); // date
RangeSlider1.max = ["NJ", "Piscataway"]; // composite array
```

See Also

Range Slider Properties, for more information on user selection editing.

MIN

Specifies the minimum slider value. If 'Single Value' is selected under the **Data** tab of the 'Range Slider Properties' dialog box, the `min` value is a single number or date (as appropriate for the data represented on the Slider). If 'Composite Value' is selected under the **Data** tab, the `min` value is an array.

Type

`Number, Date,` `or Array`

Examples

```
RangeSlider1.min = 10; // number
RangeSlider1.min = new Date("December 31, 2009"); // date
RangeSlider1.min = ["NJ", "Piscataway"]; // composite array
```

See Also

Range Slider Properties, for more information on user selection editing.

LOGSCALE

Specifies that the slider for a numerical field should use a logarithmic scale. The base of the log is determined automatically.

Type

`Boolean`	`true: Use log scale` `false: Use linear scale (default)`

Example

```
RangeSlider1.logScale = true;
```

See Also

Range Slider Properties, for more information on user selection editing.

MAXVISIBLE

Specifies whether the slider's maximum value is displayed.

Type

`Boolean`	`true: Display maximum (default)` `false: Hide maximum`

Example

```
RangeSlider1.maxVisible = false;
```

See Also

Range Slider Properties, for more information on user selection editing.

MINVISIBLE

Specifies whether the slider's minimum value is displayed.

Type

Boolean	`true: Display minimum (default)` `false: Hide minimum`

Example

```
RangeSlider1.minVisible = false;
```

See Also

Range Slider Properties, for more information on user selection editing.

RANGETYPE

Specifies the units ('Year', 'Month', 'Day') corresponding to the initial *length* of a Date Range Slider.

Type

Integer	`StyleConstant.YEAR: Units of year`
	`StyleConstant.MONTH: Units of month and year`
	`StyleConstant.DAY: Units of day, month, year`
	`StyleConstant.NUMBER: Numeric field`

Example

```
RangeSlider1.minVisible = false;
```

See Also

Range Slider Properties, for more information on user selection editing.

TICKVISIBLE

Specifies whether tick marks are displayed on the slider scale.

Type

Boolean	`true: Display tick marks (default)` `false: Hide tick marks`

Example

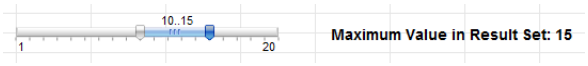
```
RangeSlider1.tickVisible = false;
```

See Also

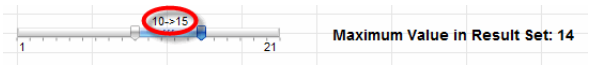
Range Slider Properties, for more information on user selection editing.

UPPERINCLUSIVE

Specifies whether the upper limit of the selection is included in the result set. For example, if the upper limit of the Range Slider selection is 15, a setting of `upperInclusive=true` (default) includes the value 15 in the result set.



A setting of `upperInclusive=false` in this case excludes the value 15 from the result set.



Note that the range display on the slider reflects the status of the `upperInclusive` property.

Type

Boolean (default)	`true: Include upper limit in selection`
	`false: Exclude upper limit from selection`

Example

```
RangeSlider1.upperInclusive=false;
```

See Also

Range Slider Properties, for more information on user selection editing.

VS.9 Calendar

This section discusses the properties that are unique to the Calendar component.

See Also

Common Properties, for properties shared with other components.
Calendar, for more information about Calendars.

DAYSELECTION

Specifies that the user can select individual days of the week (when the Calendar is in 'Monthly' view mode). Otherwise, the user must select the week in its entirety.

Type

Boolean	`true: enable individual day selection` `false: disable day selection (default)`

Example

```
RangeSlider1.daySelection = true;
```

See Also

Calendar Properties, for more information on user selection editing.

DOUBLECALENDAR

Specifies the calendar view type. The single-panel view shows one month or one year (depending on *yearView* setting), while the double-panel view shows two months or two years (depending on *yearView* setting).

Type

Boolean true: double-panel calendar
false: one-panel calendar

Example

```
Calendar1.doubleCalendar = true;
```

See Also

Calendar Properties, for more information on user selection editing.

DROPDOWN

Specifies a collapsible style for the calendar.

Type

Boolean true: collapse calendar when not in use
false: do not collapse calendar (default)

Example

```
Calendar1.dropdown=true;
```

See Also

Calendar Properties, for more information on user selection editing.

PERIOD

Specifies whether the two panels of a Calendar in *doubleCalendar* mode represent a period comparison or a simple date range.

Type

Boolean true: display date comparison
false: disable date range (default)

Example

```
Calendar1.period = true;
```

See Also

Calendar Properties, for more information on user selection editing.

YEARVIEW

Specifies whether the Calendar should be displayed in “year view” or “month view”. Year view displays one or two years (depending on *doubleCalendar* setting) and allows selections at the month level. Month view displays one or two months (depending on *doubleCalendar* setting)

and allows selections at the week or day level (depending on *daySelection* setting).

Type

Boolean true: display year view
false: disable month view (default)

Example

```
Calendar1.yearView=true;
```

See Also

Calendar Properties, for more information on user selection editing.

SINGLESELECTION

Prevents the user from selecting multiple date units on the Calendar. (This property is not supported for Viewsheets displayed using HTML, such as those on iPad.).

Type

Boolean true: enable single selection
false: disable single selection (default)

Example

```
Calendar1.singleSelection=true;
```

See Also

Calendar Properties, for more information on user selection editing.

VS.10 Text

Note: Script placed on a Text element does not re-execute when the user makes Viewsheet selections. To re-execute the script, place the script in the *onRefresh Handler*.

This section discusses the properties that are unique to the *Text* component.

See Also

Common Properties, for properties shared with other components.
Text, for more information about the Text component.

HYPERLINK

Specifies a hyperlink to a report, Viewsheet, or URL.

Type

String, or inetsoft.report.Hyperlink

Example

```
Text1.hyperlink = "Tutorial/Interactive";
```

See Also

Text Properties, for general information about the Text component.

Hyperlinks, in *Advanced Dashboard Design*, for basic information on hyperlinks.

SHADOW

Adds a drop shadow to the component. (Drop shadow for a Text component is not preserved on export.).

Type

```
Boolean       true: add drop shadow
              false: no drop shadow
```

Example

```
Text1.shadow = true;
```

See Also

Text Properties, for general information about the Text component.

TEXT

Specifies the text contents of the Text component.

Type

```
String
```

Example

```
Text1.text = "Hello World";
```

See Also

Text Properties, for general information about the Text component.

VALUE

Specifies the contents displayed by the Text component in cases when the component is bound to a data field (i.e., displaying an aggregate). If the Text component is not bound to a data field, the `value` property is undefined.

Type

```
Number
```

Example

```
if (Text1.value > 10000) {
  Text1.value = 'Maximum exceeded';
}
```

See Also

Text Properties, for general information about the Text component.

VS.11 Image

This section discusses the properties that are unique to the *Image* component.

See Also
Common Properties, for properties shared with other components.
Image, for more information about the Image component.

HYPERLINK

Specifies a hyperlink to a report, Viewsheet, or URL.

Type
`String, or inetsoft.report.Hyperlink`

Example
```
Image1.hyperlink = "Tutorial/Interactive";
```

See Also
Image Properties, for general information about the Image component.
Hyperlinks, in *Advanced Dashboard Design*, for basic information on hyperlinks.

IMAGE

Specifies an image file either by an uploaded image name (e.g., `'image1.gif'`) a resource path on the server (e.g., `'/library/image1.gif'`), a URL (e.g., "`http://www.google.com/images/logos/ps_logo2.png`"), or a BLOB stored in a data block. See *Image Properties* for information on uploading images.

Type
`String` name of image (e.g., '/myImage.jpg')

Example
```
Image1.image = '/library/image1.gif';
```

See Also
Changing Images with Script, for an approach to dynamically changing images.
getImage(string), for another approach to dynamically changing images.
Image Properties, for more information on images.

MAINTAINASPECTRATIO

When *scaleImage* is enabled, `maintainAspectRatio` specifies that the overall proportions of the image are maintained during scaling. When disabled, the image will scale to the dimensions of the component boundary using the 9-cell method (see *scale9*).

Type

Boolean	`true: keep proportions constant` `false: allow image fill component boundary`

Example

```
Image1.maintainAspectRatio = false;
```

See Also

Scaling an Image, for more information on aspect ratio.
Image Properties, for more information on images.

SCALE9

When *scaleImage* is enabled and *maintainAspectRatio* is disabled, `scale9` specifies the number of pixels on each image boundary that should be left unscaled, in the order `[top left bottom right]`. This permits you to scale only the interior of an image, leaving edge treatments unmodified.

Type

Array	`Pixels: [top left bottom right]`

Example

```
Image1.scaleImage = true;
Image1.maintainAspectRatio = false;
Image1.scale9 = [3,2,4,4];
```

See Also

Image Properties, for more information on images.
Scaling an Image, for more information on 9-cell scaling.

SCALEIMAGE

Specifies that the image should be scaled to the Image component dimensions. Scaling can be set to either preserve the image proportions (*maintainAspectRatio*, default) or to resize just a selected portion of the image (*scale9*).

Type

Boolean	`true: scale image to container` `false: keep original image size`

Example

```
Image1.scaleImage = true;
```

Note that the *scaleImage* and *tile* properties are mutually exclusive.

See Also

Image Properties, for more information on images.

SHADOW

Adds a drop shadow to the component.

Type

```
Boolean       true: add drop shadow
              false: no drop shadow
```

Example

```
Image1.shadow = true;
```

See Also

Image Properties, for more information on images.

TILE

Specifies that the image should be repeated (tiled) vertically and horizontally to fill the component boundaries.

Type

```
Boolean       true: tile the image
              false: do not tile
```

Example

```
Image1.tile = true;
```

Note that the *tile* and *scaleImage* properties are mutually exclusive.

See Also

Image Properties, for more information on images.

VS.12 Gauge, Thermometer, Sliding Scale, Cylinder

This section discusses the properties that are unique to the following output components: Gauge, Thermometer, Sliding Scale, Cylinder.

See Also

Common Properties, for properties shared with other components.
Thermometer/Cylinder/Sliding Scale/Gauge, for more information about Output components.

HYPERLINK

Specifies a hyperlink to a report, Viewsheet, or URL.

Type

```
String, or inetsoft.report.Hyperlink
```

Example

```
Gauge1.hyperlink = "Tutorial/Interactive";
```

See Also

Output Component Properties, for general information about output components.
Hyperlinks, in *Advanced Dashboard Design*, for basic information on hyperlinks.

LABELVISIBLE

Specifies whether the Gauge displays a digital representation of the current reading. (This applies only to Gauge faces that support a digital display.)

Type

Boolean	true: display digital measure (default) false: hide digital display

Example

```
Gauge1.labelVisible = false;
```

See Also

Output Component Properties, for more information on Gauges.

MAJORINC

```
Note: Increments that are too large or too small may result in
undesirable Gauge appearance.
```

Specifies the interval between display of labels on the Gauge. For example, a value of 15 indicates that the gauge labels should be placed at positions of 0, 15, 30, 45, etc. The `majorInc` value should be larger than the *minorInc* value

Type

Number	Spacing between Gauge labels

Example

```
Gauge1.majorInc = 1000;
```

See Also

Output Component Properties, for more information on Gauges.

MAX

The value of the largest label displayed on the Gauge face.

Type

Number	Value of highest gauge label

Example

```
Gauge1.max = 16000;
```

See Also

Output Component Properties, for more information on Gauges.

MIN

The value of the smallest label displayed on the Gauge face.

Type

Number	Value of lowest gauge label

Example

```
Gauge1.min = -15;
```

See Also

Output Component Properties, for more information on Gauges.

MINORINC

```
Note: Increments that are too large or too small may result in
undesirable Gauge appearance.
```

Specifies the interval between minor ticks on the Gauge. For example, a value of 15 indicates that tick marks should be placed at positions of 0, 15, 30, 45, etc. The `minorInc` value should be smaller than the *majorInc* value.

Type

Number	Spacing between minor tick marks

Example

```
Gauge1.minorInc = 100;
```

See Also

Output Component Properties, for more information on Gauges.

RANGECOLORS

Specifies an array of one to three colors to be used for the Gauge ranges (color zones). The positions of the colors on the Gauge are determined by the *ranges* array. Colors can be specified by any of the methods shown for the *background* property.

Type

Array of colors	See *background* for color syntax

Example

```
Gauge1.rangeColors = ['green','yellow','red'];
Gauge1.ranges = [5,10,15];
```

See Also

Output Component Properties, for more information on Gauges.

RANGEGRADIENT

Specifies smooth blending for range color boundaries. If disabled, the color boundaries are sharp and well-defined.

Type

Boolean	true: blend colors at boundaries (default) false: do not blend colors

Example

```
Gauge1.rangeColors = ['green','yellow','red'];
Gauge1.ranges = [5,10,15]
Gauge1.rangeGradient = false;
```

See Also

Output Component Properties, for more information on Gauges.

RANGES

Specifies an array of one to three values defining the upper boundaries of the Gauge ranges (color zones). The colors corresponding to these boundaries are determined by the *rangeColors* array. To use the *max* Gauge value as the boundary for a range, simply omit the boundary value for the desired range.

Type

```
Array                Upper range boundaries
```

Example

```
// Omit 'red' boundary value to automatically use Gauge max
Gauge1.rangeColors = ['green','yellow','red'];
Gauge1.ranges = [5,10];
```

See Also

Output Component Properties, for more information on Gauges.

SHADOW

Adds a drop shadow to the component.

Type

```
Boolean       true: add drop shadow
              false: no drop shadow
```

Example

```
Gauge1.shadow = true;
```

See Also

Output Component Properties, for more information on Gauges.

VALUE

If the output component is bound to a data field (i.e., displaying an aggregate), value returns the value displayed by the component. If the output component is not bound to a data field, the value property specifies the value to display.

Type

```
Number
```

Example

```
if(Gauge1.value > 100) {
  alert('Excessive value detected.')
}
```

See Also

Text Properties, for general information about the Text component.

VS.13 Slider

This section discusses the properties that are unique to the Slider component.

See Also

Common Properties, for properties shared with other components.
Slider and Spinner, for more information about Sliders.

CURRENTVISIBLE

Specifies that the value currently selected on the Slider is numerically displayed at all times. Otherwise, the current value is numerically displayed only while the user is actively dragging the Slider.

Type

Boolean	true: display currently selected value false: hide currently selected value

Example

```
Slider1.currentVisible = false;
```

See Also

Slider and Spinner Properties, for more information on Slider settings.

INCREMENT

Specifies the interval between displayed labels on the Slider. For example, a value of 15 indicates that the Slider labels should be placed at positions of 0, 15, 30, 45, etc.

Type

Number	Spacing between Slider labels

Example

```
Slider1.increment = 15;
```

See Also

Slider and Spinner Properties, for more information on Slider settings.

LABELVISIBLE

Specifies whether the labels on the Slider scale are displayed.

Type

Boolean	true: display scale labels false: hide scale labels

Example

```
Slider1.labelVisible = false;
```

See Also

Slider and Spinner Properties, for more information on Slider settings.

MAX

The value of the largest label displayed on the Slider scale.

Type

Number	Value of highest scale label

Example

```
Slider1.max = 250;
```

See Also

Slider and Spinner Properties, for more information on Slider settings.

MAXVISIBLE

Specifies whether the Slider's maximum value is displayed.

Type

Boolean	true: Display maximum (default) false: Hide maximum

Example

```
Slider1.maxVisible = false;
```

See Also

Slider and Spinner Properties, for more information on Slider settings.

MIN

The value of the smallest label displayed on the Slider scale.

Type

Number	Value of lowest scale label

Example

```
Slider1.min = -50;
```

See Also

Slider and Spinner Properties, for more information on Slider settings.

MINVISIBLE

Specifies whether the Slider's minimum value is displayed.

Type

Boolean	true: Display minimum (default) false: Hide minimum

Example

```
Slider1.minVisible = false;
```

See Also

Slider and Spinner Properties, for more information on Slider settings.

TICKVISIBLE

Specifies whether tick marks are displayed on the Slider scale.

Type

Boolean	true: Display tick marks (default) false: Hide tick marks

Example

```
Slider1.tickVisible = false;
```

See Also

Slider and Spinner Properties, for more information on Slider settings.

VS.14 Spinner

This section discusses the properties that are unique to the Spinner component.

See Also

Common Properties, for properties shared with other components.
Slider and Spinner, for more information about Sliders.

INCREMENT

Specifies the interval between the values when a user increments or decrements the Spinner. For example, a value of 15 indicates the Spinner should traverse values of 0, 15, 30, 45, etc. (assuming that *min*=0).

Type

Number	Spinner increment

Example

```
Spinner1.increment = 20;
```

See Also

Slider and Spinner Properties, for more information on Slider settings.

MAX

The largest value that the Spinner can represent. The user will not be able to use the Spinner to enter a larger value.

Type

Number	Value of highest Spinner input

Example

```
Spinner1.max = 250;
```

See Also

Slider and Spinner Properties, for more information on Slider settings.

MIN

The smallest value that the Spinner can represent. The user will not be able to use the Spinner to enter a smaller value.

Type

Number	Value of lowest Spinner input

Example

```
Spinner1.min = -50;
```

See Also

Slider and Spinner Properties, for more information on Slider settings.

VS.15 Check Box, Radio Button, Combo Box

This section discusses the properties that are unique to the Check Box, Radio Button, and ComboBox components.

See Also

Common Properties, for properties shared with other components.
Radio Button, Check Box, Combo Box, in *Advanced Dashboard Design*, for more information about the ComboBox component.

LABLES

Specifies the labels (as an array) displayed in a Check Box, Radio Button, or ComboBox component. This property can be used only within component-level script (see *Adding Component Script*), and the *values* property must specify an array of the same length.

Type

```
Array
```

Example

```
ComboBox1.labels = ['Region','State','City'];
ComboBox1.values = [1,2,3];
```

See Also

Common Properties, for properties shared with other components.
selectedObject and *selectedLabel*, to access the current user selection.
Radio Button, Check Box, Combo Box, in *Advanced Dashboard Design*, for more information about Input components.

ROWCOUNT

Specifies the number of items displayed when a ComboBox is expanded.

Type

```
Number
```

Example

```
ComboBox1.rowCount = 10
```

See Also

Common Properties, for properties shared with other components.
Radio Button, Check Box, Combo Box, in *Advanced Dashboard Design*, for more information about Input components.

VALUES

Specifies the values (as an array) for a Check Box, Radio Button, or ComboBox component. This property can be used only within component-level script (see *Adding Component Script*), and the *lables* property must specify an array of the same length.

Type

```
Array
```

Example

```
ComboBox1.labels = ['Region','State','City'];
ComboBox1.values = [1,2,3];
```

See Also

Common Properties, for properties shared with other components.
selectedObject and *selectedLabel*, to access the current user selection.
Radio Button, Check Box, Combo Box, in *Advanced Dashboard Design*, for more information about Input components.

VS.16 TextInput

This section discusses the properties that are unique to the TextInput component.

See Also

Common Properties, for properties shared with other components.
TextInput Properties, in *Advanced Dashboard Design*, for more information about the TextInput component.

VALUE

Specifies the contents displayed by the TextInput component.

Type

```
String or Number
```

Example

```
if (TextInput1.value > 10000) {
  TextInput1.value = 'Maximum exceeded';
}
```

See Also

Common Properties, for properties shared with other components.
TextInput Properties, for general information about the TextInput component.

VS.17 Submit Button

This section discusses the properties that are unique to the Submit component.

See Also

Common Properties, for properties shared with other components.
Submit Button Properties, in *Advanced Dashboard Design*, for more information about the Submit Button component.

REFRESHAFTERSUBMIT

When `true`, refreshes the Viewsheet after submission. This is useful when you want Viewsheet components to be immediately updated with the submitted data (for example, when the new data is submitted from an editable table to the database).

Type

Boolean	`true: refresh the Viewsheet after submission` `false: do not refresh the Viewsheet`

Example

```
refreshAfterSubmit = true;
```

See Also

Common Properties, for properties shared with other components.
Submit Button Properties, in *Advanced Dashboard Design*, to set 'Refresh Viewsheet after Submit' in the dialog box.

VS.18 Upload

This section discusses the properties that are unique to the Upload component.

See Also

Common Properties, for properties shared with other components.
Upload Properties, in *Advanced Dashboard Design*, for more information about the Upload component.

GETFILENAME()

Returns the full server path to a file uploaded on the server using the *Upload* component.

Example

```
if (Upload1.isLoaded())) {
  alert('File uploaded to ' + Upload1.getFileName())
}
```

The returned path and filename can be used, for example, to write the file to a database. See *Committing User-Modified Data to Database (Database Write-Back)* for information on how to construct a script that writes to a database.

See Also

Common Properties, for properties shared with other components.
Upload Properties, for general information about the Upload component.

ISLOADED()

Returns `true` if a file has been successfully uploaded by the Upload component.

Example

```
if (Upload1.isLoaded())) {
  alert('File uploaded to ' + Upload1.getFileName())
}
```

When using the Upload component without an accompanying 'Submit' button, enable the 'Submit on Change' option in the 'Upload Properties' dialog box.

See Also

Common Properties, for properties shared with other components.
Upload Properties, for general information about the Upload component.

VS.19 Line

This section discusses the properties that are unique to the Line component.

See Also

Common Properties, for properties shared with other components.
Creating a Shape, for more information about Line.

BEGINARROWSTYLE

Specifies the style of arrow to place at the beginning of the line.

Type

Integer 0: No arrow
StyleConstant.ARROW_LINE_1: Solid arrow
StyleConstant.ARROW_LINE_2: Stick arrow
StyleConstant.ARROW_LINE_3: Outline arrow

Example

```
Line1.beginArrowStyle = StyleConstant.ARROW_LINE_3;
```

See Also

Shape Component Properties: Shape Tab, for more information on Line settings.

ENDARROWSTYLE

Specifies the style of arrow to place at the end of the line.

Type

Integer 0: No arrow
StyleConstant.ARROW_LINE_1: Solid arrow
StyleConstant.ARROW_LINE_2: Stick arrow
StyleConstant.ARROW_LINE_3: Outline arrow

Example

```
Line1.endArrowStyle = StyleConstant.ARROW_LINE_3;
```

See Also

Shape Component Properties: Shape Tab, for more information on Line settings.

LINESTYLE

Specifies the style of the line.

Type

Integer one of the available *Line Styles*

Example

```
Line1.lineStyle = StyleConstant.THIN_LINE;
```

See Also

Shape Component Properties: Shape Tab, for more information on Line settings.

SHADOW

Adds a drop shadow to the component.

Type

```
Boolean       true: add drop shadow
              false: no drop shadow
```

Example

```
Line1.shadow = true;
```

See Also

Shape Component Properties: Shape Tab, for more information on Line settings.

VS.20 Rectangle

This section discusses the properties that are unique to the Rectangle component.

See Also

Common Properties, for properties shared with other components.
Creating a Shape, for more information about Line.

LINESTYLE

Specifies the style of the line.

Type

```
Integer       one of the available Line Styles
```

Example

```
Rectangle1.lineStyle = StyleConstant.THIN_LINE;
```

See Also

Shape Component Properties: Shape Tab, for more information on Rectangle settings.

ROUNDCORNER

Specifies the radius of corner curvature in pixels.

Type

```
Integer       corner curvature (pixels)
```

Example

```
Rectangle1.roundCorner = 30;
```

See Also

Shape Component Properties: Shape Tab, for more information on Rectangle settings.

SHADOW

Adds a drop shadow to the component.

Type

```
Boolean       true: add drop shadow
              false: no drop shadow
```

Example

```
Rectangle1.shadow = true;
```

See Also

Shape Component Properties: Shape Tab, for more information on Rectangle settings.

VS.21 Oval

This section discusses the properties that are unique to the Oval component.

See Also

Common Properties, for properties shared with other components.
Creating a Shape, for more information about Line.

LINESTYLE

Specifies the style of the line.

Type

```
Integer       one of the available Line Styles
```

Example

```
Oval1.lineStyle = StyleConstant.THIN_LINE;
```

See Also

Shape Component Properties: Shape Tab, for more information on Oval settings.

SHADOW

Adds a drop shadow to the component.

Type

```
Boolean       true: add drop shadow
              false: no drop shadow
```

Example

```
Oval1.shadow = true;
```

See Also

Shape Component Properties: Shape Tab, for more information on Oval settings.

VS.22 Tabbed Interface

This section discusses the properties that are unique to the Tabbed Interface.

See Also
Common Properties, for properties shared with other components.
Creating a Tabbed Interface, for more information about Tabs.

LABELS

Specifies the text in the tab labels (as an array of Strings).

Type
Array of String tab label text

Example

```
Tab1.labels = ['Summary', 'Details'];
if (RadioButton1.selectedObject == 'City') {
  Tab1.labels[1] = 'City Details';
}
else {
  Tab1.labels[1] = 'State Details';
};
```

See Also
Tabbed Interface Properties, for more information on Tabbed Interface.

SELECTEDINDEX

Returns the index of the currently-selected tab. This property is read-only.

Type
Integer index of active tab

Example

```
if (Tab1.selectedIndex == 0) {
  Gauge1.visible = true;
} else {
  Gauge1.visible = false;
}
```

See Also
Tabbed Interface Properties, for more information on Tabbed Interface.

VS.22.1 Viewsheet

This section presents properties for an embedded Viewsheet. These properties are set in the host Viewsheet.

See Also
Nesting Viewsheets in *Advanced Dashboard Design* for information on embedding one Viewsheet within another.

ISACTIONVISIBLE(NAME)

Returns the value of the specified property for an embedded Viewsheet.

Parameters

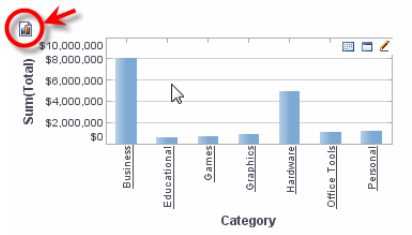
name	name of property (String): 'Open': user access to embedded Viewsheet

Example

```
if (!Viewsheet1.isActionVisible("Open")) {
  alert('Embedded Viewsheet cannot be opened.');
}
```

SETACTIONVISIBLE(NAME,BOOLEAN)

Sets the value of the specified property for an embedded Viewsheet. The '`Open`' property determines whether the embedded Viewsheet provides an 'Open' button to the end-user. (By default, this 'Open' button is visible whenever the embedded Viewsheet contains hidden components.)



Parameters

name	name of property (String): 'Open'
boolean	false: hide the 'Open' button true: show the 'Open' button

Example

```
Viewsheet1.setActionVisible("Open",false);
```

See Also

Nesting Viewsheets in *Advanced Dashboard Design* for information on embedding one Viewsheet within another.

Advanced Server Administration

Enterprise Manager is a tool for administering the server environment. Enterprise Manager allows you to perform the following tasks, among many others.

- Adjust look-and-feel, tune performance settings, server clustering, etc.
- Manage the repository files and data space.
- Start and stop the Scheduler and administer scheduled tasks.
- Administer archiving, exporting, and printing capabilities.
- Package and deploy the application or groups of assets.
- Monitor the current status of user sessions, reports, and queries.
- Administer security, security integration, users and roles, permissions, etc.

This Guide describes the Style Intelligence server environment and the features of Enterprise Manager.

1 Getting Started with Enterprise Manager

Note: Enterprise Manager does not permit path names and file names to contain Chinese or other Unicode characters.

This chapter explains how to coordinate team development with server deployment, and how to configure the server environment using the Enterprise Manager.

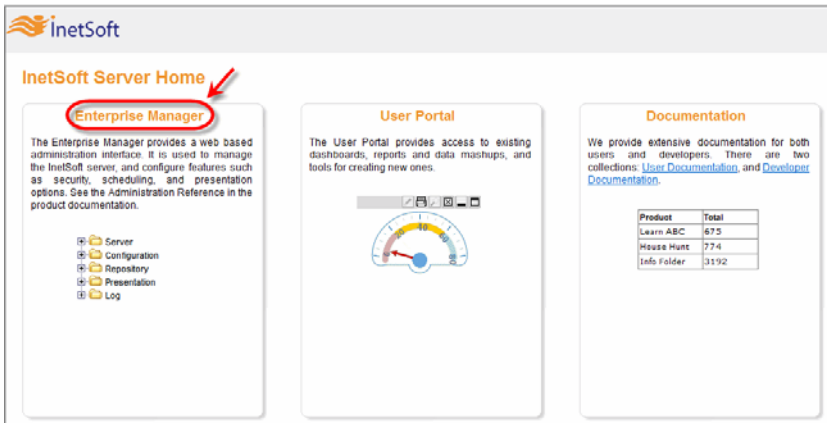
1.1 Launching Enterprise Manager

Enterprise Manager is a web-based tool that allows you to administer the report server. To launch the Enterprise Manager in a Microsoft Windows environment, follow these steps:

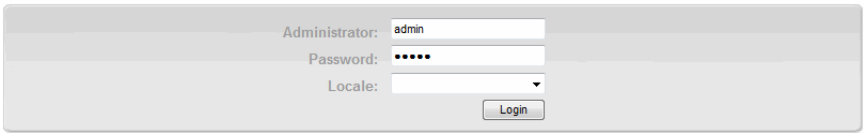
1. In the Windows *Start* menu, select *Style Intelligence* > *Style Intelligence Server.*

   This launches the provided Jetty web server on the local machine, and points the web browser to the Style Intelligence webapp homepage, default URL `http://localhost:8080/sree/`.

2. Click the 'Enterprise Manager' link. This opens the Enterprise Manager login screen.



3. Enter your login credentials, and click 'Login'.



   The default login/password is `admin/admin`. For security reasons, login information expires after five minutes. If you receive a 'Login Expired' warning, simply re-enter your credentials to log in.

If the server is already running, you can access the Enterprise Manager by entering the following URL in a web browser:

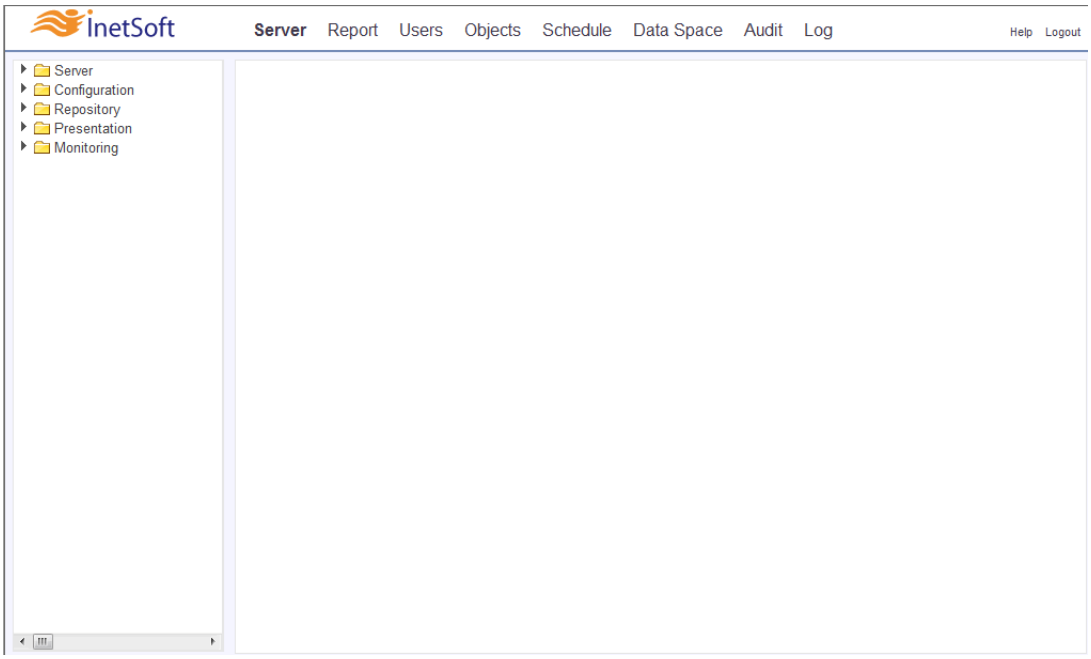
```
http://{servername}:{port}/sree/EnterpriseManager
```

In the address, `{servername}` should be the name or IP address of the machine running the administration servlet, and `{port}` should be the application's assigned port number (default: 8080). For example:

```
http://localhost:8080/sree/EnterpriseManager
```

1.2 Navigating the Enterprise Manager

After you have logged in to Enterprise Manager, use the tabs at the top of the page to access the various administrative operations.



Under each tab, use the nodes on the left-side navigation tree to set component properties. Some components have their properties split into multiple pages, e.g., 'Options' and 'Security'. You can access these pages by using the tabs at the top of the page.

Before you begin managing resources and configuring components in Enterprise Manager, verify that you have completed the following steps:

- Set up the repository directory. See the *Repository Directory* section.
- Set up the servlet repository URL. See the *Editing the Servlet and Server URL* and *Monitoring Servlet Status* sections.
- Specify valid license keys. See the *Administering License Keys* section.

- Ensure that JDK 1.6 or higher is being used, and that the web browser is JavaScript-enabled. See the *Java* and *Browser Requirements* sections.

1.3 Repository Directory

The repository directory contains all of the configuration files. The two most important configuration files for operation of the server are the following:

- **`sree.properties`**: Contains configuration information for a particular repository. File paths specified in `sree.properties` can be absolute, or relative to the repository directory. Relative paths can use the SREE Home environment variable, e.g., `$(sree.home)/repository.xml`.
- **`repository.xml`**: Contains the XML specification for the logical structure (listing of reports and folders) in the repository.

Other configuration files include the following:

- **`schedule.xml`**:Contains the XML specification for configuration of the Scheduler.
- **`datasource.xml`**: Contains the XML specification for configuration of database connections, data models, and Virtual Private Models (VPMs).
- **`query.xml`**: Contains the XML specification for SQL queries associated with particular data sources.
- **`sree.log`**: Contains log events (plain text) for the SREE server.
- **`schedule.log`**: Contains log events (plain text) for the Scheduler application.

When you use Enterprise Manager to change settings, the new settings are saved to the configuration files in the current repository directory (SREE Home).

See Also
Scheduler, for information about using the Scheduler application.

1.3.1 Specifying the Repository Directory

The default location for the repository directory is the root of the classpath (i.e., the *WEB-INF/classes* directory). To use this location, no further settings are required.

To specify a different location for the repository directory, set the context parameter `sree.home` in the `web.xml` file of the Style Intelligence webapp. For example, to use a directory called *C:/SRConfigFiles* as the repository, add the following context parameter:

```
<context-param>
```

```
    <param-name>sree.home</param-name>
    <param-value>C:/SRConfigFiles</param-value>
</context-param>
```

To use a database to store the repository files, see *Data Space*

1.3.2 Creating a Clean Repository Directory

When you first install Style Intelligence, the default repository directory is populated with a number of example data sources, queries, reports, and other assets which are provided for learning purposes. In most cases, when you begin to develop your own reports and assets, you will find it helpful to use a repository that is not cluttered with these sample materials.

It is recommended that you back up files before deleting. See *Backing up the Data Space files*.

To create a clean repository directory in the existing SREE Home location (*WEB-INF/classes*), remove all of the files currently in the SREE Home directory, except for the following two essential files:

- `dbProp.properties` (audit database settings)
- `sree.properties` (servlet configuration settings)

You may also wish to retain the following file in this directory:

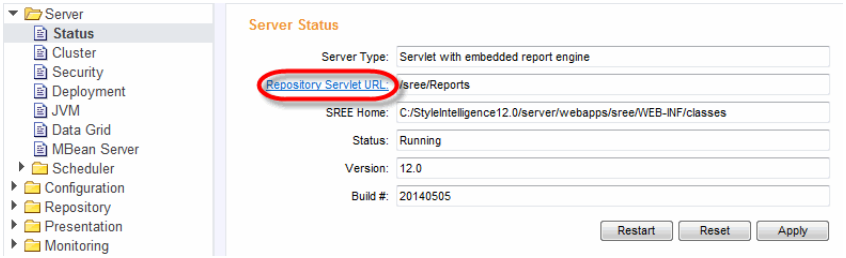
- `dashboard-layouts.xml` (required for end-users to construct dashboards)

To create a clean repository in a different location, see *Specifying the Repository Directory* to change the repository location, and add the files listed above.

1.4 Editing the Servlet and Server URL

There are two types of URLs required by the 'Server Status' page in Enterprise Manager:

- *Repository Servlet URL*:
  For 'Servlet with embedded report engine' and 'Servlet with server clustering' options on the 'Server Configuration' page.
- *Server URL*:
  For the 'RMI server' option on the 'Server Configuration' page.



Repository Servlet URL sets the base URL for resource lookup in the repository (User Portal). This URL is used to form requests that are sent to the server as a result of Portal actions.

The form of the 'Repository Servlet URL' is

```
http://{machine}:{port}/{webapp}/{URLpattern}
```

For example:

```
http://localhost:8080/sree/Reports
```

The cluster.default-Proxy property specifies a default for use when the 'Repository Servlet URL' is omitted.

For most application servers, the URL can be relative (e.g., "`/sree/Reports`") or blank, so the servlet builds its own URL. However, some application servers may not return the correct URL to the servlet. In this case, the 'Repository Servlet URL' property must be used to supply the correct URL.

Generally, the `{URLpattern}` should match the pattern specified for the Repository Servlet in the `web.xml` file. See *Setting the Servlet URL Pattern* for more details.

Server URL refers to the binding name that is used to bind the report repository object to the RMI registry. There are several formats for the URL, which allows a great deal of flexibility regarding the naming and binding of the report repository. The format is as follows:

```
//[server-name]:[port-number]/[binding name]
```

The *server name* is required while the *port number* is optional. If the port number is not specified, the default is 1099 for RMI. The *binding name* is optional, and will default to *RepletRepository* if not specified.

See Also

Configuring Server Clustering, for information on using a clustered environment.

Viewing Server Status, for general information about the 'Status' page.

1.5 Setting the Servlet URL Pattern

The Repository Servlet processes browser requests that use the URL pattern specified for the "`replets`" servlet in the `web.xml` file for the `sree` webapp. The configuration for the default pattern ("`/Reports`") is shown below.

```
<servlet>
   <servlet-name>replets</servlet-name>
   <servletclass>inetsoft.sree.web.ServletRepository</
servlet-class>
</servlet>
```

```
<servlet-mapping>
   <servlet-name>replets</servlet-name>
   <url-pattern>/Reports</url-pattern>
</servlet-mapping>
```

The Administration Servlet processes requests that use the URL pattern specified for the "`manager`" servlet in the `web.xml` file for the `sree` webapp. The configuration for the default pattern ("`/EnterpriseManager`") is shown below.

```
<servlet>
   <servlet-name>manager</servlet-name>
   <servlet-class>inetsoft.sree.adm.AdmServlet</servlet-
class>
</servlet>

<servlet-mapping>
   <servlet-name>manager</servlet-name>
   <url-pattern>/EnterpriseManager</url-pattern>
</servlet-mapping>
```

1.6 Monitoring Servlet Status

The repository servlet status is displayed on the 'Server' > 'Status' page (under the **Server** tab). The status can be 'Running' or 'Stopped'. See *Viewing Server Status* for more details.

See Also

Monitoring Server Activity, for information on viewing user reporting activity.

1.7 Servlet Environment and Prerequisites

This section discusses some requirements for server operation.

Java

Information regarding the application server's JVM and CLASSPATH can be found under the 'Server' > 'JVM' node on the tree (under the **Server** tab).

Browser Requirements

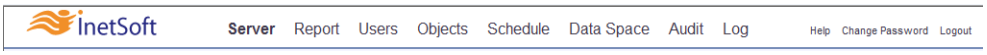
The only requirement for the Enterprise Manager and User Portal is a web browser that supports AJAX/JavaScript/Flash. This includes all modern browsers.

2 User Interface Introduction

This chapter provides a quick guide to the Enterprise Manager user interface, and explains where you can find more information about particular features.

2.1 Server Tab

This section discusses the options and features that you can access and configure under the **Server** tab. The following sub-sections correspond to the nodes on navigation tree under the **Server** tab.



2.1.1 Server Node

The 'Server' node under the **Server** tab also allows you to configure the following options:

Status

Edit the Repository Servlet URL and license keys, and refresh the server configuration settings. See *Viewing Server Status*.

Cluster

Configure server clustering. See *Configuring Server Clustering*.

Scheduler

Configure the Scheduler application. See the *Scheduler* chapter.

Security

Configure the security provider. See the *Security* chapter.

Deployment

Enable live report deployment.

JVM

Information regarding the application server's JVM and CLASSPATH.

2.1.2 Configuration Node

The 'Configuration' node under **Server** tab provides access to the following features:

Server

Configure the Style Intelligence server type.

Datasource/Lib

Specify the location of the data source files and select the connection pool type to use. See *Specifying Data Source Information*.

Database

Specify the connection parameters for the database used to store audit records and cached asset data. See *Configuring Database for Audit/Data Space*.

Mail

Control how a report is emailed to users. See *Configuring Server Email*.

Printer

Specify server printers. See *Configuring Server Printing*.

Localization

Specify the localization mappings. See *Localization/Internationalization*.

Performance

Specify performance tuning options. See *Tailoring Server Performance*.

2.1.3 Repository Node

The 'Repository' node under **Server** tab provides access to the following features:

Data Space

Configure storage for the data space (the contents of the repository directory, SREE Home), including assets. See the *Data Space* section.

Archive

Configure archive options. See the *Archive* section.

Audit

Configure audit options. See the *Auditing* section.

Cache

Specify the location of cached files, whether cached files are cleaned up at server startup, and the maximum cache life. See the *Cache* section.

Disk Quota

Specify a disk quota for users. See the *Disk Quota* section.

Export Asset

Export selected assets to an archive file (JAR) on disk. See the *Exporting Assets* section.

Import Asset

Import selected assets from an archive file (JAR) on disk. See the *Importing Assets* section.

Asset Dependency

Schedule the asset dependency audit task. See *Auditing Asset Dependencies*.

Materialized View

Optimize data retrieval performance for Viewsheets. See *Creating a Materialized View* and *Managing Materialized Views*.

2.1.4 Presentation Node

Presentation options control how reports are generated and presented to users.

General

Configure the sort order of the Repository tree, as well as formats for dates, times, and images.

Portal

Configure the overall appearance of the User Portal. See the *Portal* section. The following pages are available:

- **Look and Feel**: Configure Portal color scheme, logos, and fonts.
- **Integration**: Configure tools and navigation tabs shown in the Portal.
- **Welcome Page**: Specify an alternative Welcome page for the Portal.

Report

Configure specific Portal customization options. The following pages are available:

- **Viewer Options**: Specify options such as custom error page, report session timeout, and toolbar location. See the *Viewer Options* section.
- **Toolbar Options**: Configure visibility and ordering of toolbar buttons on generated reports. See the *Report Toolbar Options* section.
- **Toolbar Icons**: Specify custom icons for toolbar buttons and repository tree. See the *Report Toolbar Icons* section.

- **Export Menu Options**: Control availability of export formats. See the *Report Export Menu Options* section.
- **Windows and Frames**: Configure Repository tree behavior for embedded applications. See the *Windows and Frames* section.

Viewsheet

Configure Viewsheet options. The following page is available:

- **Toolbar Options**: Configure visibility and ordering of toolbar buttons on Viewsheets. See the *Viewsheet Toolbar Options* section.
- **Export Menu Options**: Control availability of export formats. See the *Report Export Menu Options* section.

Dashboard

Configure the display of dashboards in the User Portal. See *Dashboards* for more details. The following pages are available:

- **Dashboard Settings**: Enable dashboards and set tab position. See *Enabling Dashboards* for more information.
- **Dashboard Configuration**: Create and edit global dashboards and layouts. See *Creating a Dashboard* for more information.
- **Dashboard Manager**: Assign global dashboards to various roles and groups. See *Managing Dashboards* for more information.

PDF

Configure settings related to generation of archived and exported reports in PDF format. The following pages are available:

- **PDF Generation**: Configure PDF document generation options. See the *PDF Generation* section.
- **Font Mapping**: Configure mappings for CJK characters. See the *CJK Fonts* section.

2.1.5 Monitoring Node

The ‘Monitoring’ node under the **Server** tab allows you to configure the following options. See *Monitoring Server Activity* for complete information.

The ‘Monitoring Level’ menu on the ‘Settings’ page determines which of the other pages are available. A setting of at least ‘Low’ is required.

Settings

Configure the granularity at which server activity is monitored. See *Configuring Monitoring Resolution*.

Reports

View and manage executing, queued, and completed reports. See the *Reports* section.

Viewsheets

View and manage executing and open Viewsheets. See the *Viewsheets* section.

Queries

View and manage executing queries. See the *Queries* section.

Cache

View report and data cache usage. See the *Cache* section.

Requests

Requires 'Monitoring Level' set to 'High' on the 'Settings' page.

View server requests by component: Report, Viewsheet, Visual Composer. See the *Requests* section.

Users

Requires 'Monitoring Level' set to 'High' on the 'Settings' page.

View and manage current user logins. See the *Users* section.

Exceptions

Requires 'Monitoring Level' set to 'High' on the 'Settings' page.

View user and system errors. See the *Exceptions* section.

Summary

View a variety of summary statistics related to memory, disk, CPU usage, top reports, and top users. See the *Summary* section.

2.2 Report Tab

The **Report** tab allows you to manage the Repository by adding, editing or deleting folders, reports, prototypes, and Viewsheets. You can also set the Repository permission levels by using the **Security** tab.

This section describes the nodes under the **Report** tab's navigation tree.

Repository

Add and manage reports, Viewsheets, and Worksheets in the Repository. See *Configuring Reports*.

Trashcan

Contains folders and reports deleted from the archive. See *Administering Archived Reports*.

User

Manage user reports in the repository. See *Managing User Reports*.

2.3 Users Tab

The **Users** tab allows you to manage user, role, and group functions. (Security must be enabled.) This section describes the nodes under the **Users** tab's navigation tree.

Users

Manage the users and groups in the system. See *Users, Roles, and Groups*.

Roles

Manage the system roles. See *Users, Roles, and Groups*.

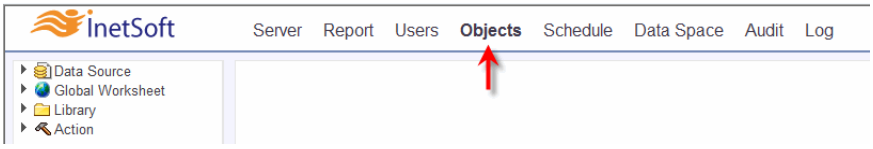
See Also

Specifying a Security Provider, for information about setting a security provider.

2.4 Objects Tab

Note: Report permissions are managed under the *Report Tab*.

The **Objects** tab allows you to manage permissions for Style Intelligence server components and resources. (Security must be enabled.) This section describes the nodes under the **Objects** tab navigation tree. See *Component/Object Permissions* for instructions on how to set permissions.



Data Source

Manage data source, data source folder, data model, query, and connection permissions.

Global Worksheet

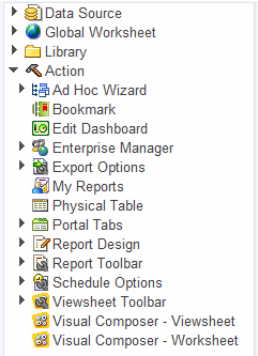
Manage Data Worksheet permissions.

Library

Manage permission for reusable components stored in the library (*stylereport.srl*): Beans, Meta-Templates, Parameter Sheets, Scripts, and Table Styles.

Action

Manage permissions for the following features:



- **Bookmark**: Permission to create new bookmarks in a Viewsheet.
- **Edit Dashboard**: Permission to edit and view dashboards displayed under the **Dashboard** tab.
- **Enterprise Manager**: Permissions for accessing Enterprise Manager. (This is generally intended for use in a multi-tenant environment; see *Design for Multi-Tenant Environment* in *Advanced Server Integration* for more details.) The following permissions can be set:
  - **Audit**: Provides access to the **Audit** tab in Enterprise Manager. (See *Audit Tab* for more information.) The user can access only audit records for the same user group.
  - **Dataspace**: Provides access to the **DataSpace** tab in Enterprise Manager. (See *Data Space Tab* for more information.)
  - **Log**: Provides access to the **Log** tab in Enterprise Manager. (See *Logging* for more information.)
  - **Objects**: Provides access to the **Objects** tab in Enterprise Manager. (See *Objects Tab* for more information.) The user can access only data sources, queries, and Worksheets for which they have permission.
  - **Report**: Provides access to the **Report** tab in Enterprise Manager. (See *Report Tab* for more information.) The user can access only reports and Viewsheets for which they have permission.
  - **Schedule**: Provides access to the **Schedule** tab in Enterprise Manager. (See *Schedule Tab* for more information.) The user can access only scheduled tasks for the same user group.
  - **Server**: Provides access to the **Server** tab in Enterprise Manager. (See *Server Tab* for more information.)
  - **Users**: Provides access to the **Users** tab in Enterprise Manager. (See *Users Tab* for more information.) The user can access only other users in the same user group.
- **Export Options**: Permission to access report export options
- **My Reports**: Permission to access ‘My Reports’ repository folder
- **Physical Table**: Permission to access physical database tables within Data Worksheets and Viewsheets
- **Portal Tabs**: Permission to access the Portal tabs: Dashboard, Report, Design, Schedule, and any custom tabs. See *Integration* for information on adding custom tabs.
- **Report Design**: Permissions pertaining to report design:

  - **Import and Export**: Permission to import and export assets from Style Studio when using the 'Server' repository option. See *Configuring a Repository*, *Deploying an Asset to a Remote Repository*, and *Importing Assets from a JAR File* for more information.
  - **Style Studio – Local**: Permission to use 'Local' execution in Style Studio when using the 'Server' repository option. See *Configuring a Repository* for more information.
  - **Style Studio – Remote**: Permission to use 'Remote' execution in Style Studio when using the 'Server' repository option. See *Configuring a Repository* for more information.

- **Report Toolbar**: Permission for report toolbar buttons
- **Schedule Options**: Permission for scheduled task actions
- **Viewsheet Toolbar**: Permission for Viewsheet toolbar buttons
- **Visual Composer – Viewsheet**: Permission for Viewsheet design
- **Visual Composer – Worksheet**: Permission for Data Worksheet design

See Also
Component/Object Permissions, for additional information.
Specifying a Security Provider, for information about setting a security provider.

2.5 Schedule Tab

The **Schedule** tab allows you to monitor and manage Scheduler tasks. See the *Scheduler* section for more information. This section describes the nodes under the **Schedule** tab's navigation tree.

Scheduled Tasks

Monitor and manage scheduled tasks. See *Monitoring and Managing Scheduler Tasks*.

Cycles

Manage Scheduler cycles. See the *Scheduler Cycle* section.

2.6 Data Space Tab

The **Data Space** tab allows you to manage all files and folders in the data space (i.e., the contents of the repository directory, SREE Home). See the *Data Space* section for more information. The navigation tree under the **Data Space** tab displays the folders and file in the data space.

Folder

Select a folder to manage the folder properties. See *Managing Folders*.

File

Select a file to manage the file properties. See *Managing Files*.

2.7 Audit Tab

If auditing is enabled, all object and data source access in Enterprise Manager and the User Portal is logged into the audit database. The **Audit** tab allows you to view the audit records. See *Auditing* for more information.

See Also

Configuring Database for Audit/Data Space, to set up a database for auditing.

2.8 Log Tab

The **Log** tab provides access to server logging features. See *Logging* for more information.

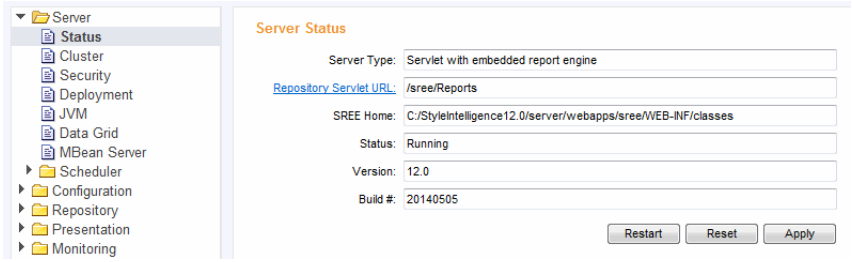
3 Server Environment

The server environment can be configured using the options available on the tree under the **Server** tab. These options configure server type and data sources, localization, and performance tuning.

3.1 Viewing Server Status

Use the 'Server' > 'Status' page under the **Server** tab to start or stop an RMI server, "restart" the server (i.e., reload the server properties file), or administer licence keys.

Figure 1. Enterprise Manager Server Status Page



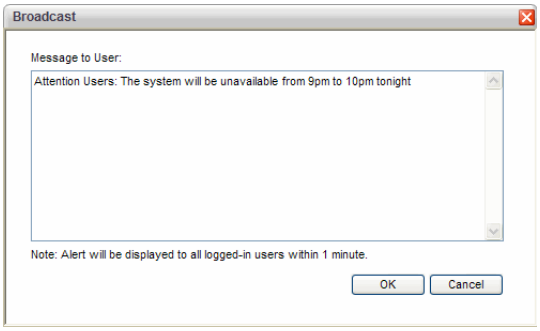
Note: The 'Restart' button does **not** restart the server. It merely refreshes the server properties by reloading the *sree.properties* file. In most cases these properties are refreshed *automatically* when *sree.properties* is modified, so there is no need to use the 'Restart' button.

The 'Repository Servlet URL' property specifies the URL used to access the repository servlet. See *Editing the Servlet and Server URL* for more information.

Broadcasting a Message to Users

The broadcast feature requires a security provider be specified.

To send an alert to all currently logged-in users, press the 'Broadcast' button. This opens the 'Broadcast' dialog box, where you can enter a message to be relayed to users.



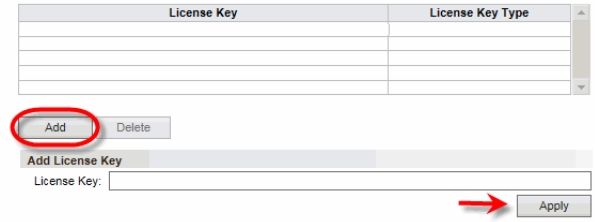
When you have entered the desired message, click ‘OK’. The message will be displayed to active users within about one minute after sending. This can be used to warn of system-wide scheduled maintenance.

Administering License Keys

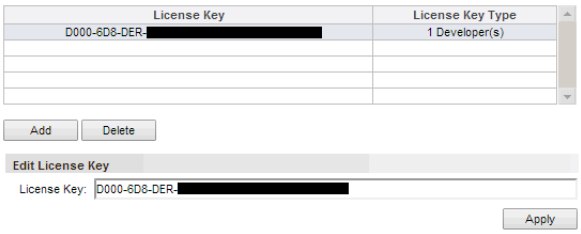
To add or modify a License Key in Enterprise Manager, open the ‘Server’ > ‘Status’ page (under the **Server** tab).

To add a new license key, follow these steps:

1. Press the ‘Add’ button.
2. Enter the new license key in the ‘License Key’ field.
3. Press the ‘Apply’ button.



To edit a existing license key, select the desired license key in the table and edit the key in the ‘License Key’ field. Press ‘Apply’.



To delete a license key, select the license key in the table, and press ‘Delete’.

See Also

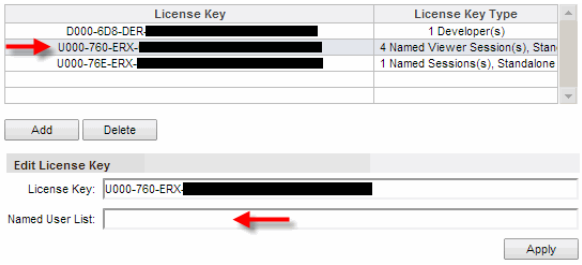
Configuring Server Clustering, for information on setting licenses in a clustered environment.

Defining Users for Named User License

To specify named users for a “Named Viewer Session” or “Named Session” license key, open the ‘Server’ > ‘Status’ page (under the **Server** tab), and follow the steps below:

1. Click on the desired “Named Viewer Session” or “Named Session” license key to select it.

2. In the ‘Named User List’ field, enter a comma-separated list of usernames.



   These are the users for which the specified license will enable a session.

3. Press ‘Apply’.

For a “Named Session” license key (which enables Visual Composer), you should specify which particular users have permission to access Visual Composer. Follow the steps below:

1. Select the **Users** tab in Enterprise Manager.

2. Assign the ‘Designer’ role to users who need Visual Composer access. See *Users, Roles, and Groups* for information on assigning user roles.

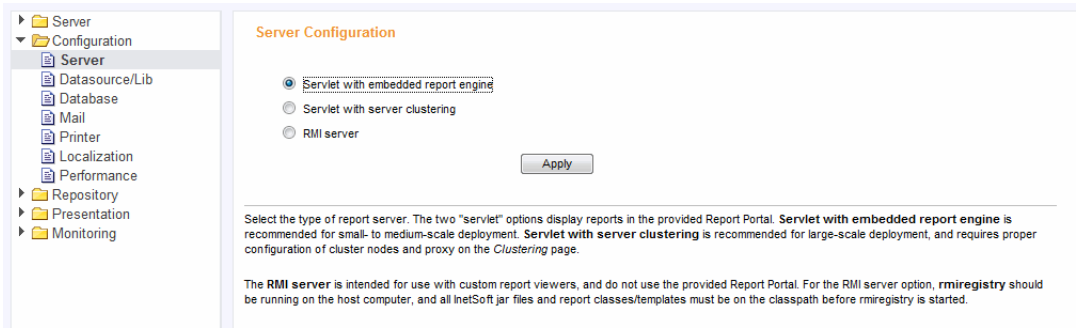
You can assign the ‘Designer’ role to more users than the maximum number allowed by the license. If the maximum number of licensed users is reached, the next named user who attempts to open Visual Composer will receive a notification that all licenses are in use.

3.2 Changing the Server Type

The replet repository is the main server component, responsible for handling all report requests from a client. Select the ‘Configuration’ > ‘Server’ node under the **Server** tab to choose one of the repository server options:

- *Servlet with Embedded Report Engine*: Servlet implementation of the repository for web-based reporting.
- *Servlet with server clustering*: Servlet implementation of the repository for a load-balanced clustering. See *Configuring Server Clustering*.
- *RMI Server*: RMI server object implementation of the repository.

Figure 2. Choose Server Type



3.2.1 Servlet with Embedded Report Engine

The ‘Servlet with embedded report engine’ option is the most commonly used configuration of the reporting engine. In this configuration, clients access the report repository via a servlet which has the reporting engine embedded within it. With this configuration, users can view all the reports and Viewsheets within a web browser (User Portal).

3.2.2 RMI Server

To run a custom client viewer on a different machine than the server, you need to set up a server object to handle the viewer requests. The default server object implementation of the repository is based on the standard Java RMI distributed object protocol.

To start an RMI repository on the server, follow these steps:

1. Start the Java Remote Object Registry process (`rmiregistry`). On a Windows platform, start the registry in a separate window as follows:

   ```
   start rmiregistry
   ```

   On a Unix platform, start the Registry as a background process:

   ```
   nohup rmiregistry &
   ```

   If the default port of 1099 is in use, you can pass a different port as a parameter.

2. Once a registry process is running, bind the RMI repository object to the registry: On the ‘Server Configuration’ page, change the server type to ‘RMI Server’, and click ‘Apply’.

3. On the ‘Server Status’ page, click the ‘Start’ button to start the RMI server.

You can also start the RMI server manually. With the `rmiregistry` running, type the following at the command line:

```
java –Dsree.home=/usr/mysreehomedir inetsoft.sree.rmi.
RemoteRepository –url //your-host-name/RepletRepository
```

You can stop and restart the RMI server using the ‘Stop’ and ‘Restart’ buttons on the ‘Server Status’ page.

3.3 Configuring Server Clustering

Server clustering is the standard method for improving the scalability of a server system. Clustering uses multiple server instances to process client report requests, thus enhancing performance over the single-server configuration.

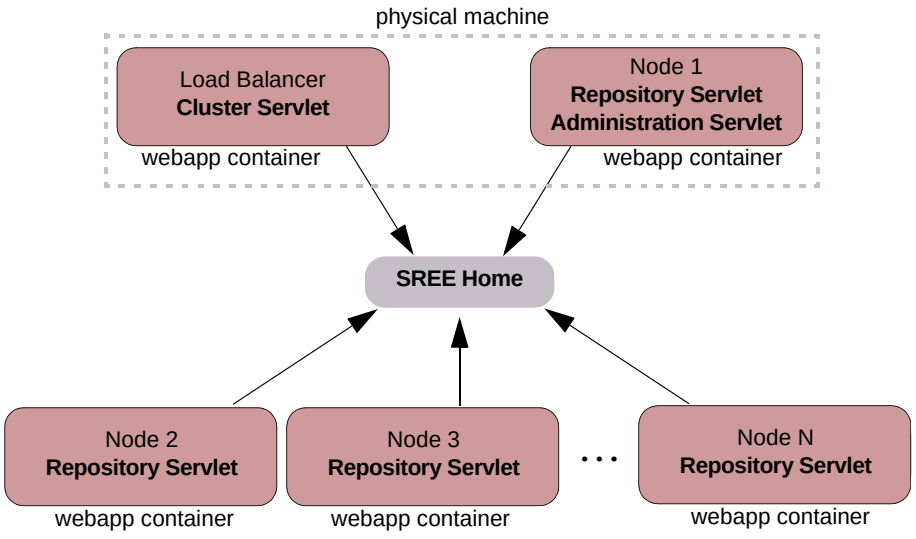
Server clustering is transparent to the end-user. The end-user interacts with a single server (the same as in a non-clustered environment), and a load balancing application automatically redirects the server requests to available cluster nodes to achieve maximum throughput.

3.3.1 Server Clustering Procedure

Clustering support in Enterprise Manager is based on server-side load distribution. A Proxy/Load-Balancer application serves as the sole interface to all clients, and manages the communication to individual server instances (cluster nodes).

The recommended cluster configuration is shown below. Note that the Load Balancer can be installed on the same physical machine as Node 1 if desired.

Figure 3. A Load Balanced Cluster



To set up a clustered environment, follow the steps below:

1. Install **Node 1**, containing the Repository servlet and Administration servlet. See *Installing Node 1 for Repository and Admin Servlets*.

2. Install **Node 2** to **Node N**. See *Installing Additional Nodes for Repository Servlets*.

3. Install the **Load Balancer**. See *Installing the Load Balancer Proxy*.

4. Define the cluster nodes in Enterprise Manager. See *Defining the Cluster Nodes*.

All cluster nodes share a common repository directory (SREE Home) which is specified as a context parameter in the deployment descriptor file (`web.xml`). To specify server properties for specific machines in the cluster, see *Assigning Machine-Specific Properties*.

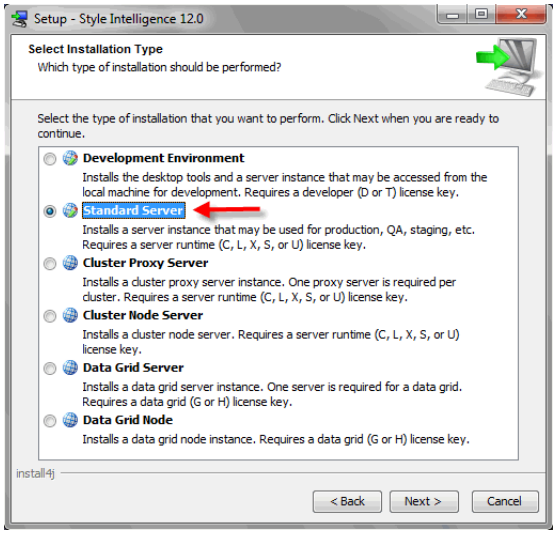
See Also
Distributed Materialization in a Clustered Environment, for details on Viewsheet materialization.

3.3.2 Installing Node 1 for Repository and Admin Servlets

Node 1 will run both a Repository servlet and an Administration servlet. Follow the steps below to configure this node:

1. Install the Style Intelligence server onto Node 1.

   Use the appropriate installation procedure for your platform. For example, on a Windows computer, run the installer executable, and select the ‘Standard Server’ option on the ‘Installation Type’ screen.



This will install the entire server framework, including the Repository and Administration servlets.

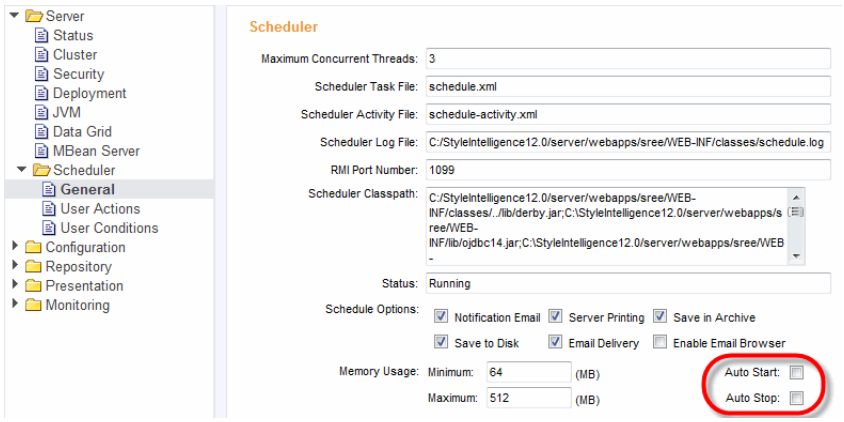
2. Launch the Style Intelligence Server on Node 1, and log into Enterprise Manager.

3. Under the **Server** tab, open the 'Server' > 'Security' page. Configure your security provider as desired. (See *Specifying a Security Provider* for full details.)

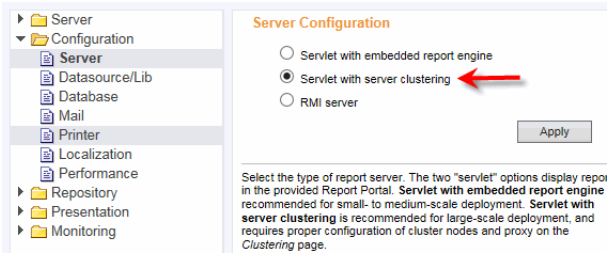
4. Open the 'Server' > 'Status' page. If required, change the 'Repository Servlet URL' to "`/sree/Reports`" and click 'Apply'. (Ignore any warning messages. See *Viewing Server Status* for full details.)

See *Configuring the Scheduler* for full details about the Scheduler application. Only one Scheduler application may run within a cluster.

5. Open the 'Server' > 'Scheduler' > 'General' page. Deselect the 'Auto Start' and 'Auto Stop' options. (See *Using a Clustered Environment* to manually run the Scheduler on a cluster node.)



6. Open the ‘Configuration’ > ‘Server’ page. Select ‘Servlet with server clustering’.



The ‘Server’ > ‘Cluster’ page is only displayed if the selected server type supports clustering. See *Changing the Server Type*.

7. Open the ‘Server’ > ‘Cluster’ page. Make the following settings:

   a. Set the ‘Node Servlet URI’ to “`/sree/Reports`”.

   b. Set the ‘Enterprise Manager URL’ to the URL of Enterprise Manager on Node 1 (i.e., the URL you are currently using to access Enterprise Manager.) For example:

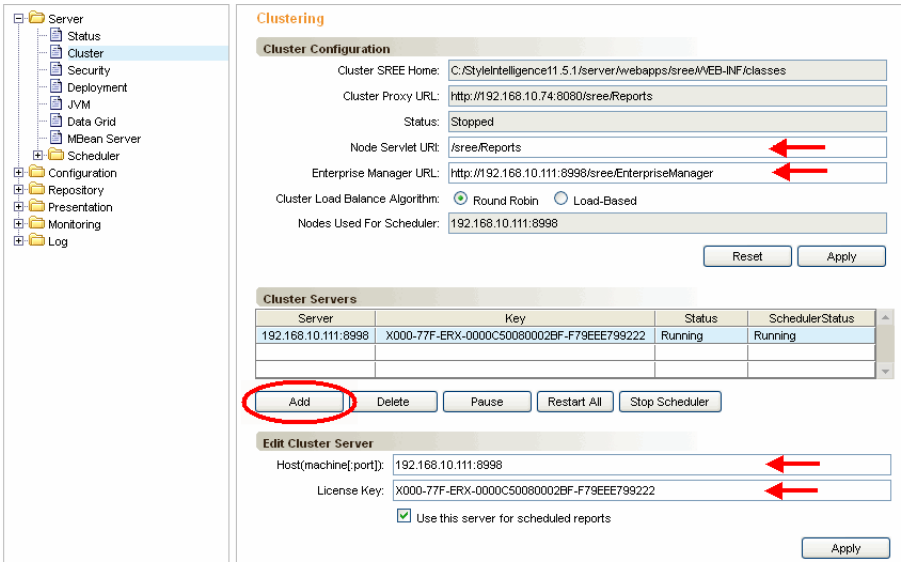
      ```
      http://node1:8080/sree/EnterpriseManager
      ```

   c. Select the ‘Cluster Load Balance Algorithm’. *Round Robin* assigns requests to the registered nodes in sequence, while *Load-based* assigns requests based on node’s load.

   d. Under the ‘Cluster Servers’ panel, click the ‘Add’ button.

You will add the other cluster nodes here at a later stage.

   e. Add Node 1 as a cluster node: Enter the host name and port (e.g, “`node1:8080`”), and a valid license key.



f. Select ‘Use this server for scheduled reports’ if you want scheduled reports to be processed this machine (in addition to on-demand reports).

8. Click the ‘Apply’ button.

9. Log out of Enterprise Manager.

10. Stop the Node 1 application server. Open the `sree.properties` file (located in the *WEB-INF/classes* directory), and make the following changes:

    a. For the properties `scheduler.classpath` and `log.output.file`, replace any occurrence of the variable `$(sree.home)` with a hard-coded absolute path to the *WEB-INF/classes* directory (SREE Home). For example:

    ```
    C\:/Program Files/Style Intelligence/server/webapps/
    sree/WEB-INF/classes/
    ```

    b. Add the following new property:

    `replet.cache.directory`=`{`*`Path to SREE Home`*`}/temp`

    For example:

    ```
    replet.cache.directory=C\:/Program Files/Style
    Intelligence/server/webapps/sree/WEB-INF/classes/temp
    ```

    c. Save your changes to the `sree.properties` file.

You may use the existing location on Node 1 if this will be accessible to all nodes (via sharing).

11. Copy the entire *WEB-INF/classes* directory to a shared location that will be accessible to all cluster nodes.

    Alternatively, you can also store configuration files and report templates in a database that can be accessed by cluster nodes. See *Configuring the Data Space*.

12. Add the following entry to the `web.xml` file of the SREE webapp (in *sree/WEB-INF*) on Node 1.

    ```
    <context-param>
       <param-name>sree.home</param-name>
       <param-value>{Path to shared location}</param-value>
    </context-param>
    ```

    For `{`*`Path to shared location`*`}` above, substitute the full path of the *WEB-INF/classes* directory that you copied in the previous step.

Any changes that you make in the Enterprise Manager (e.g., enabling/ disabling security) after the cluster is set up and started will only be reflected when the nodes are restarted.

3.3.3 Installing Additional Nodes for Repository Servlets

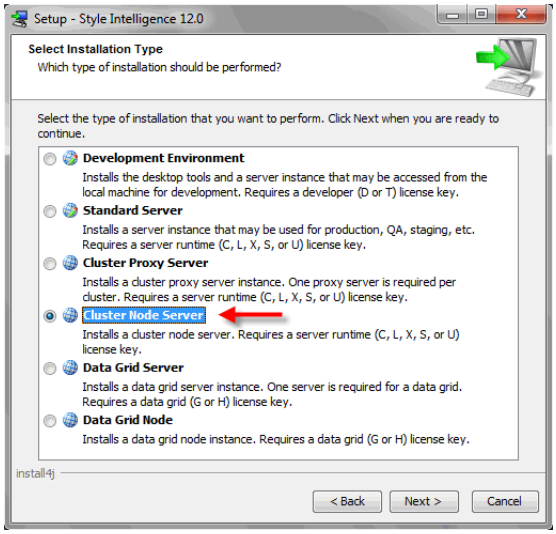
Node 2 through Node N should each run only the Repository servlet (`inetsoft.sree.web.ServletRepository`). For each Cluster Node machine, configure the node by following the steps below:

1. Run the Style Intelligence installer on the machine that you want to host the Cluster Node.

```
Each cluster node must have an unique license key.
```

2. On the 'License Key' page, enter the appropriate key.

3. On the 'Installation Type' page, select the 'Cluster Node Server' option.

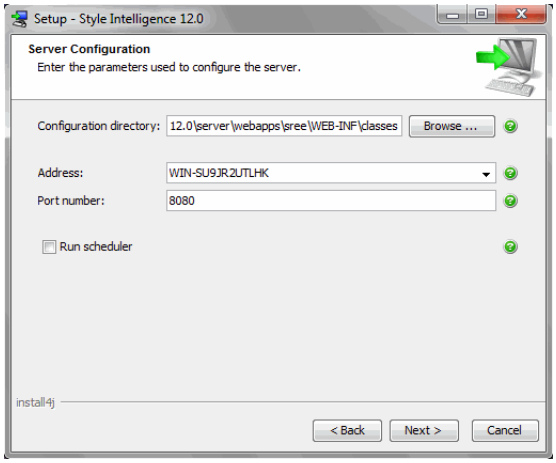


4. On the 'Select Destination Directory' screen of the installer, set the 'Destination directory' to the identical installation path used on Node 1 (e.g., "`C:/Program Files/Style Intelligence...`").

   Use this same installation location for every node. This allows the hard-coded paths in the `sree.properties` file to be resolved to the local machine, thus preventing an inefficient transfer of log and cache files across the network.

5. On the 'Server Configuration' page, set the 'Shared registry directory' to the SREE Home directory that you shared during the installation of Node 1. (See *Installing Node 1 for Repository and Admin Servlets*.) Set the 'Address' and 'Port number' as

appropriate for the machine onto which you are installing the Cluster Node.



6. Follow the rest of the on-screen installation instructions to complete the installation of the Cluster Node.

3.3.4 Installing the Load Balancer Proxy

This proxy can also be used for single sign-on purposes. See *Configuring Single Sign-On (SSO)* in *Advanced Server Integration* for more details.

The *Proxy* or *Load-Balancer* is a lightweight application (servlet) that forwards requests from clients to server nodes. The proxy obtains its settings from a copy of the `sree.properties` file. This copy can be shared with the cluster nodes (as described below) or stored independently (for example, in case the cluster is running outside the company firewall and cannot access a shared drive).

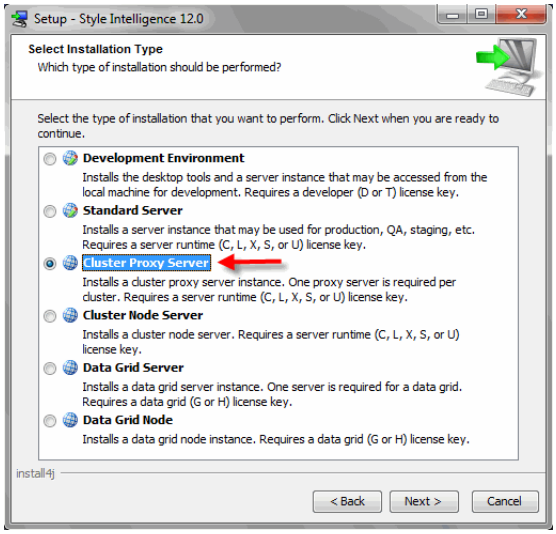
The Load Balancer node uses a different servlet than the other nodes. The Cluster servlet (`inetsoft.sree.web.ClusterServlet`) runs within the SREE webapp. It is recommended that the Cluster servlet be the only servlet deployed on its web server so that the process is lightweight and not prone to failures from other applications in the same web server.

Follow the steps below to configure the Load Balancer:

1. Run the Style Intelligence installer on the machine that you want to host the Load Balancer.

2. On the ‘License Key’ page, enter the appropriate key.

3. On the ‘Installation Type’ page, select the ‘Cluster Proxy Server’ option.

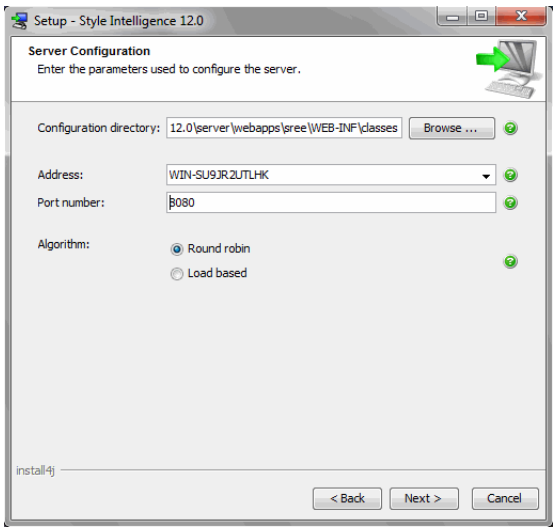


4. On the ‘Server Configuration’ page, make the following settings:

   a. Set the ‘Shared registry directory’ to the SREE Home directory that you shared during the installation of Node 1. (See *Installing Node 1 for Repository and Admin Servlets*.)

   b. Set the ‘Address’ and ‘Port number’ as appropriate for the machine onto which you are installing the Cluster Proxy.

   c. Select the cluster load balance ‘Algorithm’. *Round Robin* assigns requests to the registered nodes in sequence, while *Load-based* assigns requests based on node’s load. You can change this later from the ‘Server’ > ‘Cluster’ page of Enterprise Manager.



5. Follow the rest of the on-screen installation instructions to complete the installation of the Cluster Proxy.

3.3.5 Defining the Cluster Nodes

Once you have completed the operations in *Installing Node 1 for Repository and Admin Servlets*, *Installing Additional Nodes for Repository Servlets*, and *Installing the Load Balancer Proxy*, you must start up the cluster and define each cluster node on the 'Clustering' page of Enterprise Manager. Follow the steps below.

1. Start the server on Node 1.
2. Start the servers on Node 2 through Node N.
3. Start the Load Balancer (Proxy) node.
4. Launch Enterprise Manager on Node 1.

```
Each cluster node must have an unique license key.
```

5. Open Enterprise Manager on Node 1. On the 'Server' > 'Cluster' page, add Node 2 through Node N to the 'Cluster Servers' panel if not already listed.
6. If you want a node to be used by the Scheduler to process scheduled reports, select 'Use this server for scheduled reports'. See *Using a Clustered Environment* for instructions on running the Scheduler.

To manage a node, select it from the list of existing cluster nodes. You can prevent new sessions from being sent to the node by pausing it (click the 'Pause'/'Resume' button). Then you can wait until the existing users logoff. This is useful if a node requires maintenance, and you do not wish existing users to be negatively impacted. You can remove the node by clicking the 'Delete' button. To reload the `sree.properties` file for all nodes, click the 'Restart All' button.

3.3.6 Assigning Machine-Specific Properties

The nodes in a cluster all share a common server repository (see *Server Clustering Procedure*), and hence all share a common properties file, *sree.properties*, located in the common repository.

To individually assign server properties to specific machines in a cluster, follow the steps below:

1. Add the property `sree.local.properties` to the common *sree.properties* file. This property should specify the location of a local properties file on each machine. For example:

   ```
   sree.local.properties = C:\myLocal.properties
   ```

2. Create a corresponding properties file with the specified name and location on each machine, e.g., *C:\myLocal.properties*.

When each server in the cluster loads its properties, it will first load properties from the common *sree.properties* file, and then load any additional properties from its own local properties file, e.g., *C:\myLocal.properties*.

3.3.7 Using a Clustered Environment

To access the User Portal when using a clustered environment, point the browser to the Load Balancer URL or to any cluster node URL (e.g., `http://node1:8080/sree/Reports`). The Portal's appearance and function is the same as in a single-server environment, but all user requests are transparently redirected to the Load Balancer.

Load Balancer

The Load Balancer distributes the requests to the different cluster nodes based on the balancing algorithm. It also continuously monitors the status of the cluster nodes at a regular interval. If a node becomes unresponsive, the Load Balancer detects the failure, removes the node from the node list, and stops forwarding requests to it. On recovery, the node is detected once again, and assigned new requests. The Load Balancer keeps track of sessions so that once a request is forwarded to a node, all subsequent requests from the same client will be forwarded to the same node. To view the activity of various nodes in the cluster, see *Monitoring Server Activity*.

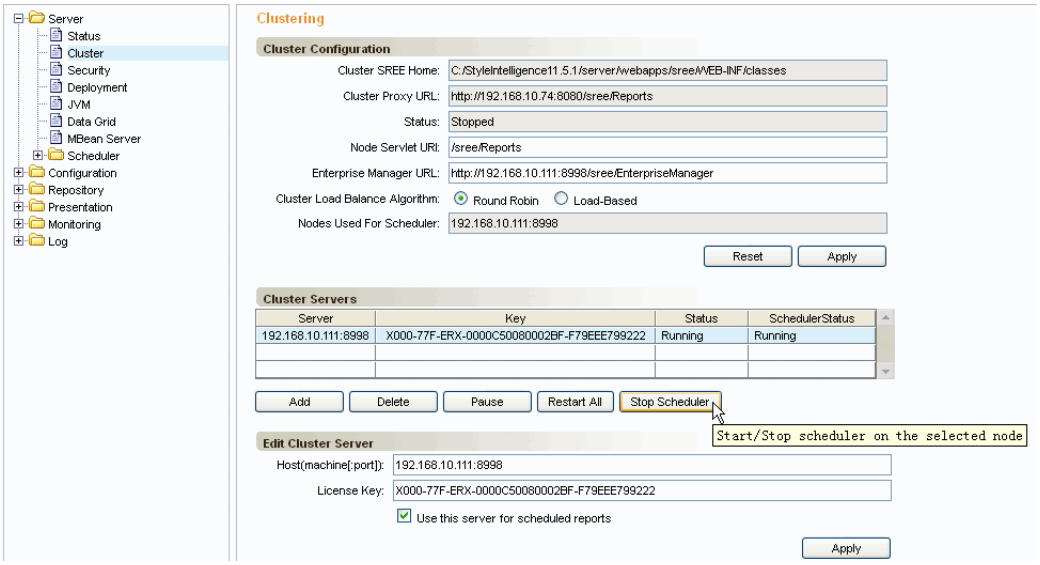
Running the Scheduler on a Cluster Node

The Scheduler application can run on only one node within the cluster at a given time. To run the Scheduler on a cluster node, follow the steps below:

1. Open Enterprise Manager on Node 1, and open the 'Server' > 'Cluster' page.

2. In the 'Cluster Servers' table, select the node on which you want the Scheduler application to run.

3. Press the 'Start Scheduler' button to start the Scheduler on the selected node.

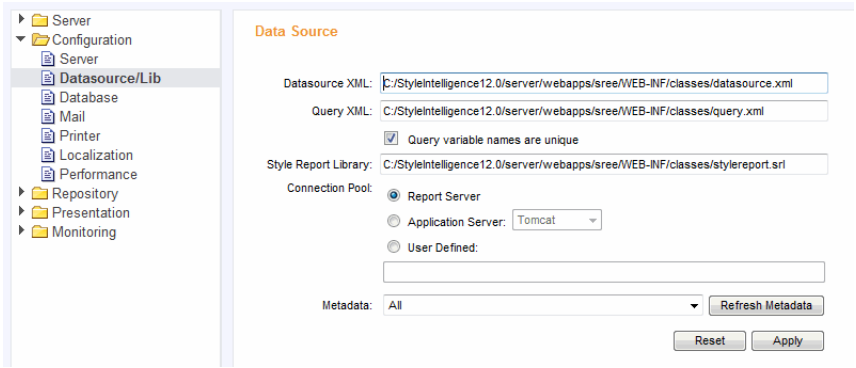


To stop the Scheduler on a given node, select the node in the table and press the 'Stop Scheduler' button.

3.4 Specifying Data Source Information

The 'Configuration' > 'Datasource/Lib' page allows you to set the location of the following repository files:

- **`datasource.xml`**: Database connection information, data model definitions, and VPM configuration.
- **`query.xml`**: Query definitions.
- **`stylereport.srl`**: Reusable report components, such as parameter sheets, table styles, script functions, beans, etc.



The 'Query variable names are unique' option allows you to specify whether a variable name that is used in multiple queries or binding conditions should be treated as unique or non-unique. If unique, the report or Viewsheet accepts a single value to be re-used across all variable occurrences (and the user is prompted only once for this value). Otherwise,

the report or Viewsheet accepts a distinct value for each occurrence (and the user is prompted once for each occurrence).

If you make a change to your database schema, select the corresponding data source in the 'Metadata' menu, and click 'Refresh Metadata'. (Select 'All' to refresh metadata for all listed data sources.) This will allow schema changes to be reflected in the listing under the 'TABLE' node of the Visual Composer **Asset** panel.

Connection Pool

The query engine uses connection pooling for enhanced database performance. The 'Connection Pool' option allows you to specify the type of connection pool to use for managing connections to JDBC data sources. Use the 'Application Server' connection pool if your webapp server is running other applications (in addition to the *sree* webapp) that also require database connections. Otherwise, use the 'Report Server' connection pool to allow Style Intelligence to manage the connections.

Supported application servers are Tomcat, WebSphere, and WebLogic.

If you select the 'Application Server' connection pool, each data source name that you define in Style Studio must match a corresponding JNDI lookup name in your application server.

Note also that the 'Application Server' connection pool option will not apply to the Scheduler, as the Scheduler runs in an independent JVM and cannot access the application server connection pool. (See the *Scheduler* section for more details.)

The default size of the pool is five connections. For enterprise level deployment, the number of connections can be increased to a more appropriate size by setting the property `jdbc.connection.pool.size` in *sree.properties*.

```
jdbc.connection.pool.size=50
```

If you select a 'User Defined' connection pool, you must enter the fully qualified class name of the `inetsoft.uql.jdbc.ConnectionPool` implementation in the provided text field, *and the class must be accessible from the web application's classpath.* See *Pooling Database Connections Programmatically* in *Advanced Server Integration* for more information.

See Also

Pooling Database Connections Programmatically, in *Advanced Server Integration*, for finer grained control over pooling.

Creating a New JDBC Data Source, in *Advanced Data Modeling*, to configure and name a data source.

3.5 Configuring Database for Audit/Data Space

Note: You must configure a database before enabling auditing.

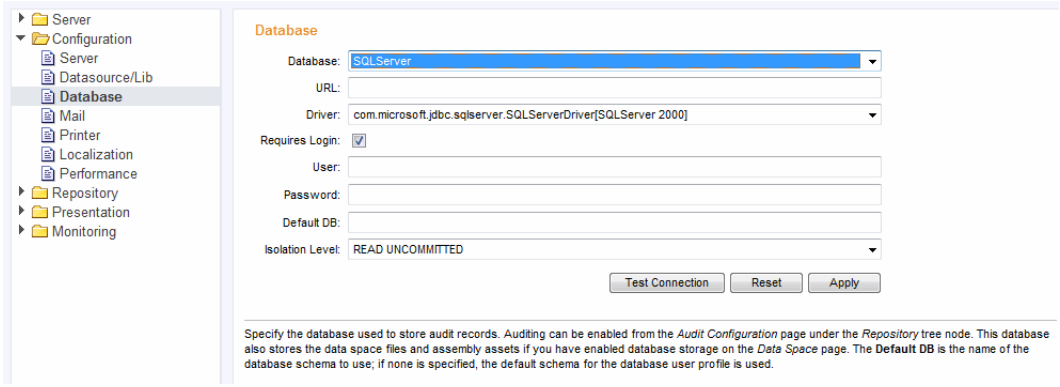
You can use a database for persistent storage of the following items:

- Data space (report templates, `datasource.xml`, `query.xml`, etc.)
- Asset repository (*asset.dat*)
- Audit Records

To configure storage for these elements, use the 'Repository' > 'Data Space' and 'Repository' > 'Audit' pages. (See the *Data Space* and *Auditing* sections for more information.)

To configure a database for persistent storage, open the 'Configuration' > 'Database' page (under the **Server** tab), and specify the database type, connection information, and the transaction isolation level.

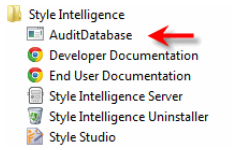
Figure 4. Database Setup Page




```
Note: Cache and temporary files cannot be stored in a database.
```

The 'Default DB' field indicates the name of the database schema to be used. If left unspecified, the default database schema for the particular user profile will be used. When you have entered the required information, test the connection by clicking on the 'Test Connection' button.

The following databases are supported for data space and auditing purposes: Oracle, DB2, SQL Server, PostgreSQL, MySQL, and Derby. For your convenience, a sample Derby database is included with the default installation. To run this database, select 'Audit Database' from the Style Intelligence program group in the Windows 'Start' menu.



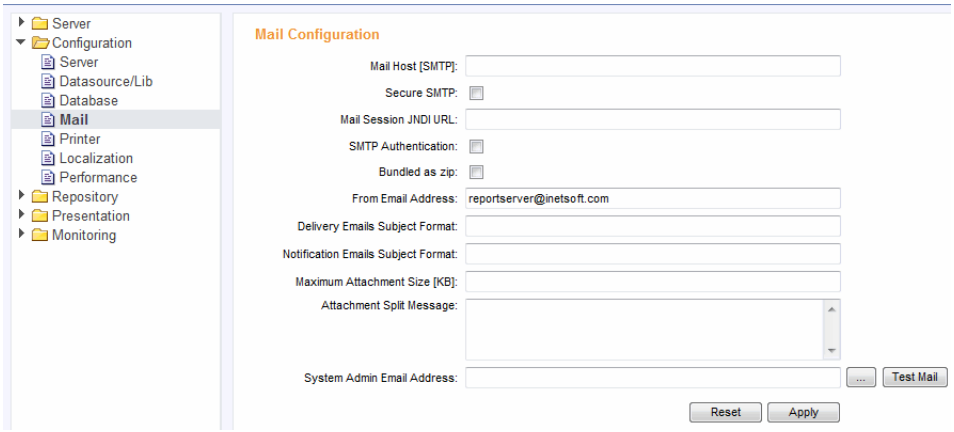
3.6 Configuring Server Email

The ‘Mail Configuration’ page specifies how the report server mails reports to users, and how end users can contact an administrator in case an error is generated during report generation.

The ‘Mail Host’ option is required, and must specify one or more valid SMTP servers, separated by commas. If the servers require secure protocol, enable the “Secure SMTP” option. The ‘From Email Address’ specifies the fixed email address from which all emails are sent.

Additional properties are described below.

Figure 5. Mail Configuration Page



Mail Session JNDI URL

If the optional ‘Mail Session JNDI URL’ property is set, the report engine will first try to obtain a session from the JNDI directory when sending emails. If this fails, it will create a new session based on the setting provided for the ‘Mail Host’ property.

Note that the *Scheduler* cannot access this ‘Mail Session JNDI URL’ property, since the Scheduler runs as a separate application independent of the application server.

System Admin Email Address

The ‘System Admin Email Address’ specifies an email address to be displayed on report error pages. A user can click the displayed email address to send a message to the administrator, alerting them to the problem.

Emails Subject Format

The ‘Delivery Emails Subject Format’ and ‘Notification Emails Subject Format’ use `java.text.MessageFormat` syntax with two parameters, the

first (index 0) being the report alias, and the second (index 1) being the date and time of report generation. For example, a report with alias ‘testReport’ and the following Subject Format specification,

```
Report named {0}, generated at {1,time} on {1,date}
```

would produce an email subject line such as the following:

```
Report named testReport, generated at 10:34:54 AM on Nov 5,
2007
```

where the time and date shown would be the actual time and date of report generation. See *Filename Parameters* for some examples of date formatting using `java.text.SimpleDateFormat`. If no alias is specified for the report in the ‘Alias’ field of the **Report** tab, then parameter “`{0}`” defaults to the report name.

The global ‘Delivery Emails Subject Format’ setting is overridden by the ‘Subject’ specified (if any) under the **Schedule** > **Action** tab of the Enterprise Manager and the User Portal.

Maximum Attachment Size

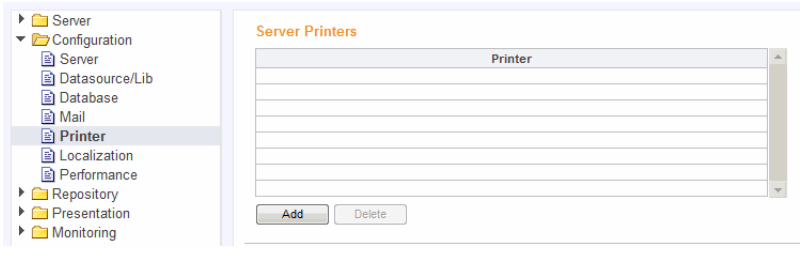
The ‘Maximum Attachment Size [kB]’ sets an upper limit for the size of an exported report file sent as a single attachment. If exceeded, the file is split into multiple files (binary split) and delivered via multiple emails. It is the responsibility of the recipient to concatenate all the fragments into one file. This concatenation can be done via command-line tools such as “type” (Windows) or “cat” (Linux). For example:

```
type file1.pdf file2.pdf > file3.pdf
```

3.7 Configuring Server Printing

The ‘Server Printers’ page allows you to configure server-side printers for use by the Scheduler and User Portal. Printers must be accessible to the server, and the printer name should be entered in exactly the same way as it appears in the system (case-sensitive). All detected printers are automatically listed in the ‘Add Printer’ menu.

Figure 6. Server Printers Page



See Also
Scheduler Actions, for information on scheduled printing.

3.8 Localization/Internationalization

Localization is a feature that allows Style Intelligence to adapt to different language environments, and is implemented for three contexts:

- User Portal interface
- Reports
- Viewsheets

When users log into the User Portal or administrators log into Enterprise Manager they can specify a locale (language-country combination), based on which Style Intelligence dynamically modifies the interface (Portal tabs, Enterprise Manager tabs, repository tree) as well as report and Viewsheet elements (text elements, chart axis titles, table header columns, etc.).

Two independent mapping files are required for localization:

- The "`srinter`" mapping file controls localization for interface text in the User Portal and Enterprise Manager (tabs, tree nodes, etc.).
- The "`SreeBundle`" mapping file controls localization for report and Viewsheet text (text in tables, charts, etc.) and data model/query names.

The following sections discuss the contents of these two files, and explain how to register a new locale.

3.8.1 Registering Locales

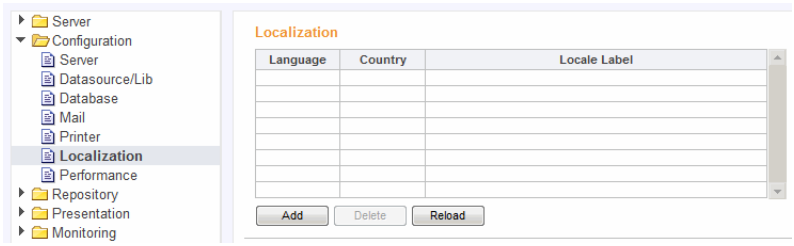
To implement localization at the Portal-level, report-level, or Viewsheet level, first register the desired locales in the 'Localization' page of the Enterprise Manager:

1. Under the **Server** tab, select 'Configuration' > 'Localization'.
2. Click 'Add' to open the 'Add Locale' panel.
3. Enter the ISO-standard two-letter codes for the desired language and country.

   These codes should match the codes used in the "`srinter`" and "`SreeBundle`" mapping file names. For example, for a French Canadian locale, enter "`fr`" for 'Language' and "`CA`" for 'Country'. (See *Localizing the User Portal Interface* and *Localizing Reports, Viewsheets, Data Models/Queries* for more details.)
4. Click 'Apply' to register the locale.

The locale entries you add in the table of the ‘Localization’ page will be displayed in the ‘Locale’ menu of the User Portal and Enterprise Manager login screens. Create a separate pair of mapping files for each locale you wish to register. See *Localizing the User Portal Interface* and *Localizing Reports, Viewsheets, Data Models/Queries* for information about how to create the mapping files.

Figure 7. Localization Page



After you create the mapping files, you can press the ‘Reload’ button below the table to reload the mappings. However, to ensure that the localizations apply correctly to all assets (including assets that were assigned Text IDs prior to creation of the mapping files), it is recommended to restart the server.

3.8.2 Localizing the User Portal Interface

Localizable aspects of the User Portal and Enterprise Manager interfaces include the text in tab elements and nodes of the repository tree and dialog box labels. The mapping (translation) file that specifies the text substitutions for these interface elements should be given the following name,

```
srinter_{language}_{country}.properties
```

where `{language}` and `{country}` are the ISO-standard two-letter codes for language and country. For example, the localization mapping file for Canadian French speakers should be called ‘`srinter_fr_CA.properties`’. All interface localization (“`srinter`”) files should be placed in the following directory:

```
sree\WEB-INF\classes\inetsoft\util
```

If the appropriate mapping file for the user’s locale selection cannot be found at runtime, the default ‘`srinter.properties`’ file is used and no localization is performed.

Each line of text in the “srinter” mapping file should have the following simple format:

```
{interface_element}={translated_interface_element}
```

The `{interface_element}` text in most cases is just the text of the actual interface element, for example, '`Design`' to represent the text displayed in the **Design** tab. Spaces in the `{interface_element}` text must be escaped by a backslash, for example, '`My\ Reports`'.

The `{translated_interface_element}` represents the localized text that you want to appear in the interface element. Spaces in the `{translated_interface_element}` do not need to be escaped.

Example: Interface Localization

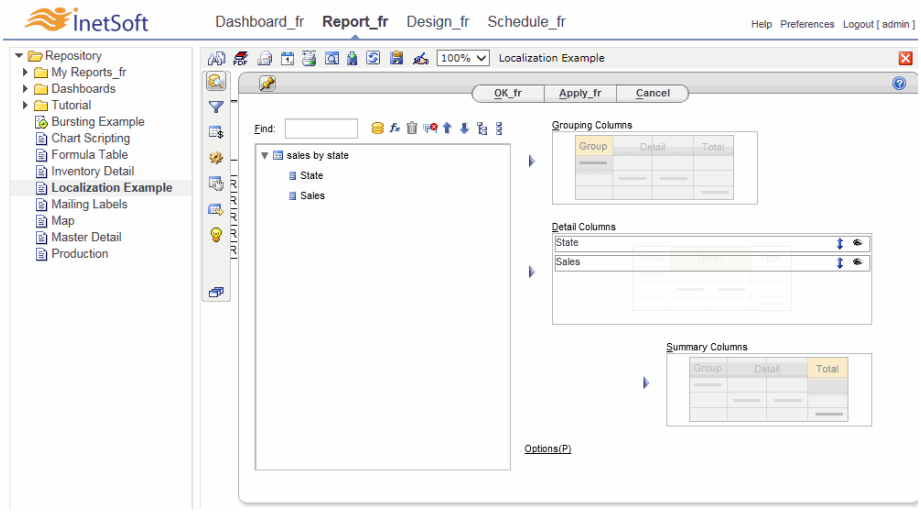
For example, the French Canadian localization file mentioned above might contain the following lines of text, where, for purposes of illustration, "`Report_fr`" is used as the French translation of "`Report`," and so on.

```
Report = Report_fr
Dashboard = Dashboard_fr
Design = Design_fr
Schedule = Schedule_fr

My\ Reports = My Reports_fr
Tutorial = Tutorial_fr

Apply = Apply_fr
OK = OK_fr
Close = Close_fr
```

Figure 8. Localized User Portal Interface



Note: The report shown in the example is developed in *Localizing Reports, Viewsheets, Data Models/Queries*.

The above mapping renames the main tabs, the Repository tree folders, and buttons with the specified (mock) French text. Consult the default '`srinter.properties`' file (available upon request) for a complete listing of modifiable interface elements.

¢

3.8.3 Localizing Reports, Viewsheets, Data Models/Queries

You can use a simple text mapping to localize most text in a report or Viewsheet, as well as data source names, data model names (entity/attribute), and query (column) names.

The first step in setting up localization for a particular report or Viewsheet is to assign Text IDs to each text element to be mapped. You can do this in the 'Localization' dialog box of Style Studio (for reports) and the 'Options' dialog box of Visual Composer (for Viewsheets). See *Localization/Internationalization* in *Advanced Report Design* and *Localization/Internationalization* in *Advanced Dashboard Design* for more information.

After you have assigned Text IDs to report and Viewsheet elements, you can create the mapping file ("resource bundle") that specifies the text substitutions for these elements. The mapping file should be given the following name,

```
SreeBundle_{language}_{country}.properties
```

where `{language}` and `{country}` are the ISO-standard two-letter codes for language and country. For example, the localization mapping file for Canadian French speakers should be called '`SreeBundle_fr_CA.properties`'. All report/Viewsheet localization ("`SreeBundle`") files must be available on the classpath, preferably in the *sree/WEB-INF/classes* directory. If the appropriate mapping file for the user's locale selection cannot be found at runtime, no localization is performed.

Similar to the "`srinter`" files for Portal interface localization (see *Localizing the User Portal Interface*), each line of text in the report/Viewsheet localization file has the following format:

```
{element_id}={translated_element_id}
```

The `{element_id}` is the Text ID assigned to the report or Viewsheet element in Style Studio or Visual Composer. Spaces in the `{element_id}` must be escaped by a backslash, for example, '`Head\ 1`'.

The `{translated_element_id}` is the translated text that you want to appear in the corresponding report or Viewsheet element. Spaces in the `{translated_element_id}` do not need to be escaped.

For example, the French Canadian localization file mentioned above, '`SreeBundle_fr_CA.properties`', might contain the following lines of text, where, for purposes of illustration, "`State_fr`" is used as the French translation of "`State`," and so on:

```
title=Localized Title_fr
```

```
state\ header=State_fr
sales\ header=Sales_fr
```

The above mapping translates the text corresponding to three Text IDs, (which might be assigned in a report or Viewsheet) into the specified mock French text:

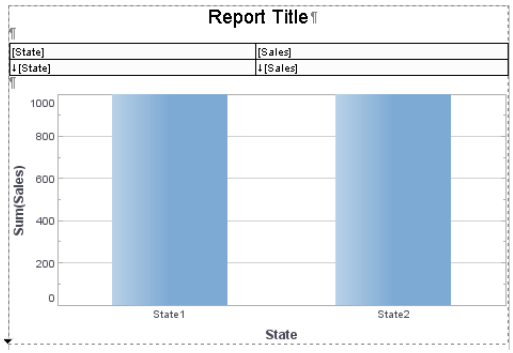
TEXT ID	TRANSLATED TEXT
title	Localized Title_fr
state header	State_fr
sales header	Sales_fr

The following example illustrates the use of a similar mapping that also translates data source names.

Example: Report Element Localization

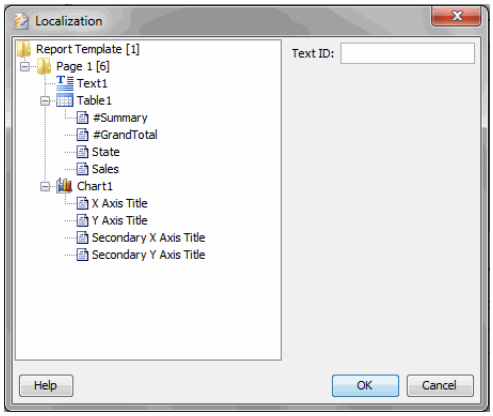
In this example, you will configure a report for French Canadian localization.You will also localize a data source name and a query name.

1. Open Style Studio, and create a blank tabular report.
2. Add a text element with the text ‘Report Title’. Format the text as desired.
3. Add a table, and bind it to the ‘Orders’ > ‘sales by state’ query.
4. Add a bar chart, and bind it also to the ‘Orders’ > ‘sales by state’ query. In the **Chart** tab of the ‘Data Binding’ dialog box, add ‘State’ on the X-axis and ‘Sales’ on the Y-axis.



You now have a report with three elements, and will assign Text IDs to several of the text components provided by those elements.

5. Open the ‘Localization’ dialog box from the Style Studio ‘Report’ menu.
6. Expand the component tree to view the text components which can be assigned Text IDs.



7. Enter the following IDs in the 'Text ID' field for the corresponding components.

```
text1: report title
State: state header
Sales: sales header
X Axis Title: X
Y Axis Title: Y
```

   Leave the Text ID box empty for the remaining elements. (See the *Localization/Internationalization* section of the *Advanced Report Design* for more information about adding Text IDs.)

8. Save the report, and deploy it to the Repository under the name 'Localization_Exp'.

You now have a deployed report with elements configured for localization. To enable localization to take place when the report is viewed, you must create the resource bundle that translates the assigned Text IDs into the desired text for the French Canadian locale.

9. Create a text file called '`SreeBundle_fr_CA.properties`', and insert the following text:

```
report\ title=Localized Report_fr
state\ header=State_fr
sales\ header=Sales_fr
X=Sales By State_fr
Y=Revenue_fr

Orders=Orders_fr
All\ Sales=All Sales_fr
```

   Note that the final two lines of the mapping file specify a translation for the 'Orders' data source and the 'All Sales' query.

10. Save the file in the *sree/WEB-INF/classes* directory.

With the report elements assigned text IDs, and the resource bundle file appropriately named and placed on the classpath, you now need to register the locale in the Enterprise Manager.

See *Localization/Internationalization* for details on registering locales.

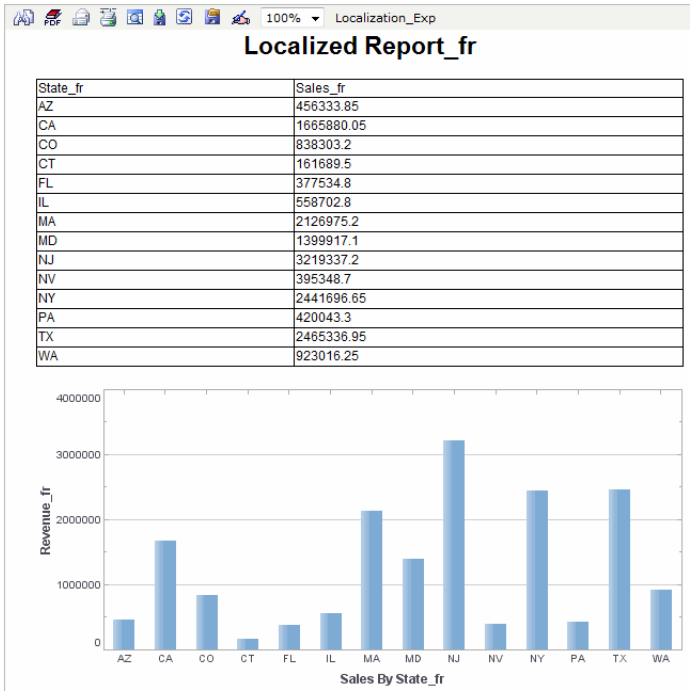
11. In the ‘Localization’ page of the Enterprise Manager, add a new locale.

    a. Specify ‘`fr`’ and ‘`CA`’ for the language and country codes, respectively.

    b. Set the ‘Locale Label’ to “French (Canada)”.

When the user logs in and specified ‘French (Canada)’ in the ‘Locale’ login menu, they will view the ‘Localization_Exp’ report with the desired localization applied. The report title, table column headings, and chart labels have all been translated as specified.

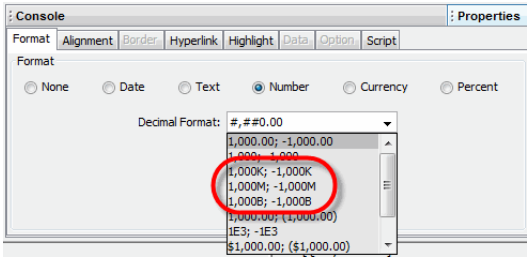
Figure 9. Report with localization applied.



¢

3.8.4 Defining Custom Number Format Multipliers

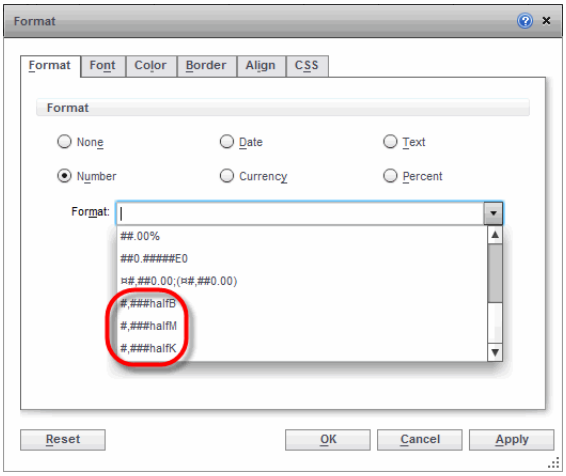
When applying a number format, users can add a suffix “K”, “M”, or “B” to the pattern string to automatically divide the numerical value by 10^3, 10^6, and 10^9, respectively.



You can define custom symbols and multipliers by adding the desired entries to the *userformat.xml* file in the server repository (*WEB-INF/classes* by default). For example, the following *userformat.xml* file defines three new multipliers, “halfK”, “halfM”, and “halfB”, that divide the numerical values by $5x10^2$, $5x10^5$, and $5x10^8$, respectively.

```
<?xml version="1.0" encoding="utf-8"?>
<userFormat>
  <NumberFormat suffix="halfK" multiplier="500"/>
  <NumberFormat suffix="halfM" multiplier="500000"/>
  <NumberFormat suffix="halfB" multiplier="500000000"/>
</userFormat>
```

When you restart the server with this *userformat.xml* file in place, the new suffixes will be available for use with number format patterns, and will be displayed among the format presets for reports and Viewsheets:



The default suffixes can be redefined but not removed.

To redefine the existing “K”, “M”, or “B” suffixes, simply assign these suffixes new values in the *userformat.xml* file.

```
<NumberFormat suffix="K" multiplier="..."/>
<NumberFormat suffix="M" multiplier="..."/>
<NumberFormat suffix="B" multiplier="..."/>
```

See Also

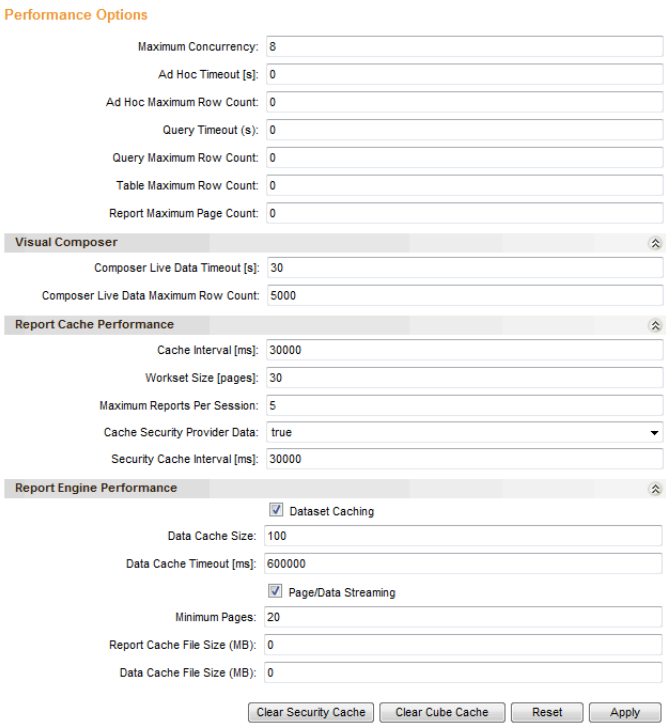
Number Format, in *Advanced Dashboard Design*, for information about formatting a Viewsheet.

Number Formats, in *Advanced Report Design*, for information about formatting a report.

3.9 Tailoring Server Performance

Server performance options are available on the 'Configuration' > 'Performance' page under the **Server** tab. These options primarily control how data is cached. (See the *Cache* section for information on setting cache location and automatic clean-up.)

Figure 10. Performance Options Page



3.9.1 Performance Options and Safeguards

The options on the 'Performance' page should be set according to the server's memory size and the performance characteristics of the application.

- **Maximum Concurrency**: The 'Maximum Concurrency' property specifies the maximum number of sessions allowed at any given time. Concurrency is the number of simultaneous user sessions that the report engine will support. A user session is initiated when the user logs in, and terminates when the user logs out or when all the reports associated with the user's session time out.
- **Query Timeout**: The 'Query Timeout' setting specifies the global timeout (seconds) for all queries during report/Viewsheet runtime

execution. The timeout setting may not be honored by all databases. The corresponding `sree.properties` entry is `query.runtime.timeout`.

- **Query Maximum Row Count**: The 'Query Maximum Row Count' setting specifies the global maximum size for all queries during report/Viewsheet runtime execution. The maximum row count setting may not be honored by all databases. The corresponding `sree.properties` entry is `query.runtime.maxrow`.
- **Table Maximum Row Count**: The 'Table Maximum Row Count' setting specifies the global maximum size for all tables during report/Viewsheet runtime execution. The corresponding `sree.properties` entry is `table.output.maxrow`. This property controls the maximum size of the final displayed table, following any grouping and aggregation.
- **Report Maximum Page Count**: The 'Report Maximum Page Count' setting specifies a global maximum on the number of pages for reports during runtime execution. The corresponding `sree.properties` entry is `report.output.maxpages`. You can override this global setting for a particular report by setting a 'Max Number of Pages' value in the 'Report Properties' dialog box of Style Studio. See *Report Properties* in *Advanced Report Design* for more information.
- **Composer Live Data Timeout**: The 'Composer Live Data Timeout' setting specifies the maximum amount of time allowed for 'Live Data' requests in Visual Composer. The timeout setting may not be honored by all databases. The corresponding `sree.properties` entry is `query.preview.timeout`. See *Showing Live Data in a Table* in *Advanced Data Mashup* for more information.
- **Composer Live Data Maximum Row Count**: The 'Composer Live Data Maximum Row Count' setting specifies the maximum number of rows retrieved for 'Live Data' requests in Visual Composer. The maximum row count setting may not be honored by all databases. The corresponding `sree.properties` entry is `query.preview.maxrow`. See *Showing Live Data in a Table* in *Advanced Data Mashup* for more information.
- **Cache Interval**: The 'Cache Interval' specifies the elapsed time (ms) between cache swaps. At the specified interval, the server checks, swaps, and prunes cached pages according to their age and access time.
- **Workset Size**: A "workset" is a group of report pages that can be swapped as a single unit between RAM and disk. A large workset size provides for more efficient disk swapping. However, a large workset size may also delay initial report availability because the report engine assembles the full workset before it begins streaming pages for display.
- **Maximum Reports Per Session**: The 'Maximum Reports Per Session' option specifies how many reports a session can open and keep active. If

a session contains more active reports than the threshold, the older reports are destroyed to conserve memory.

- **Cache Security Provider Data**: The 'Cache Security Provider Data' option, when set to 'true', enables the caching of user, role, and permission information. This is recommended.
- **Security Cache Interval**: The 'Security Cache Interval' option specifies the interval (in milliseconds) after which the cache of user, role, and permission information is automatically cleared.
- **Dataset Caching**: The 'Dataset Caching' option specifies whether query results are cached and shared across reports and/or Viewsheets. If the option is disabled, then the query results are only shared within the same report and/or Viewsheet if multiple elements are bound to the same query.
- **Data Cache Size**: The 'Data Cache Size' option specifies the number of datasets that can be cached and reused.
- **Data Cache Timeout**: The 'Data cache timeout' controls the persistence (ms) of a dataset in the data cache. If the dataset is not used for the specified time period, it is removed from the cache. The default timeout is 30 seconds.
- **Paging Threshold**: The 'Paging Threshold' setting controls the initiation of dataset paging. If a dataset exceeds the threshold, it is paged to disk in order to conserve memory.
- **Page/Data Streaming**: The 'Page/Data Streaming' option allows reports to be processed in parallel with data loading. If the 'Page/Data Streaming' option is checked, the engine begins processing the report when data becomes partially available, resulting in quicker report display. Users may see a partially completed report marked with a "*" in the page total, indicating that additional pages are still being generated.
- **Minimum Pages**: The 'Minimum Pages' option specifies the number of pages to generate before initiating page streaming. If this value is less than the workset size (see above), streaming will begin only after the first workset has been fully generated. However, you can enter a value of "1" to force the first page of the report to display immediately, regardless of workset size. For small reports that have fewer pages than the minimum, streaming will be disabled so that end users do not see a partially generated report.
- **Clear Security Cache**: The 'Clear Security Cache' button clears the cache of user, role, and permission information. This is useful if security information has changed and you wish to manually refresh the server cache. (Note: The cache is automatically refreshed at the interval specified by the 'Security Cache Interval'.)

- **Clear Cube Cache**: The ‘Clear Cube Cache’ button deletes the cached OLAP cube structure. This is useful in the event that the cache becomes corrupted, which might occur if the application server fails to terminate properly. (An error such as “Locking failed due to time out” is a possible symptom of this.)

See Also
Appendix B.4, *Web Properties and Safeguards*, for property descriptions.

3.9.2 Limiting Query Size and Execution Time

Several properties are available to control the size of queries returned during preview and runtime operation. These properties provide important safeguards that prevent end-users from erroneously requesting very large amounts of data while designing Worksheets.

The first group of properties below can be configured from the ‘Performance’ page in Enterprise Manager. See *Performance Options and Safeguards* above.

```
query.runtime.timeout
query.runtime.maxrow
table.output.maxrow
report.output.maxpages
query.preview.maxrow
query.preview.timeout
query.adhoc.maxrow
query.adhoc.timeout
asset.sample.maxrows
query.estimate.enabled
assetData.thread.count
```

See Appendix B.4, *Web Properties and Safeguards*, for a description of each property.

See Also
Performance Options and Safeguards, for Enterprise Manager properties.
Precautions and Safeguards, in *Advanced Report Design*, for report-level safeguards.
Limiting the Number of Rows in a Table, in *Advanced Data Mashup*, for Worksheet-level safeguards.

3.10 Monitoring Server Activity

The ‘Monitoring’ node under the **Server** tab allows you to monitor and manage a variety of aspects of server activity. On these pages, you can view reports, Viewsheets, and queries that are currently executing on the server, and can terminate undesired activity. You can also view currently active users, and terminate any undesired sessions.

Style Intelligence monitoring features utilize *managed bean* (MBean) technology, which allows you to access monitoring information from a

remote computer. See *Configuring Remote Monitoring* for more information on accessing the MBean information.

3.10.1 Configuring Monitoring Resolution

The ‘Settings’ page under the ‘Monitoring’ node allows you to make global adjustments to the resolution of monitoring activity. The following options are available.

A ‘Monitoring Level’ of at least ‘Low’ is needed for basic access

Table 3. Report Execution

Monitoring Level	Determines the resolution with which server events are monitored, ‘Low’, ‘Medium’, or ‘High’. Data The level required for monitoring a given property is noted in the following sections.
Sample Period	The duration in seconds over which monitoring metrics (e.g., throughput) should be aggregated.
Data Set Size	The number of data points that should be displayed on the charts of the ‘Summary’ page.
Refresh Interval	The time in seconds between monitoring page updates. A shorter value will keep the displays more current.

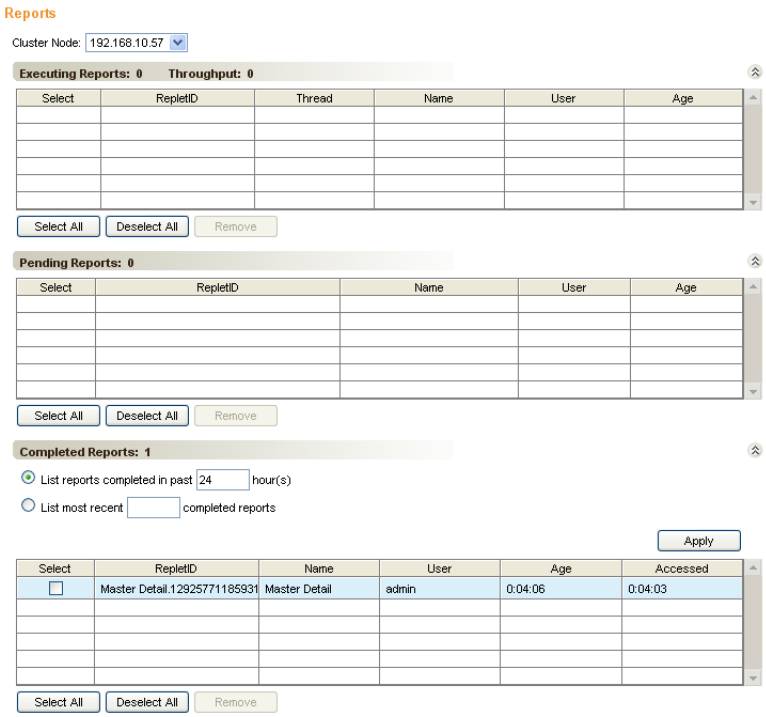
3.10.2 Reports

The ‘Reports’ page under the ‘Monitoring’ node provides key information about reports that are currently executing on the server, reports that are awaiting execution, and reports that have already been executed.

A *pending* report is a report that the user has requested, but that has not yet been assigned an execution thread (due to natural or designed processing limitations). As soon as the required processing resources become available, the pending report will be generated and immediately displayed to the user.

A *queued report* is a report that the user has requested, but for which they explicitly selected ‘Add to Queue’ from the report loading screen. The queued report continues to execute, but is not automatically displayed to the user.

In a clustered environment, select a machine from ‘Cluster Node’ menu to view the monitoring statistics for that machine.



The page provides the following information:

Table 4. Report Execution

Replet ID	A unique identifier for the report.
Thread	The thread within which the report is executing. Click the link to obtain a stack trace.
Name	The name of the report.
User	The user who executed the report.
Pages	The number or report pages that have been processed. (Requires ‘Monitoring Level’ set to ‘Medium’ on the ‘Settings’ page.)
Age	The time elapsed since report generation began.

To terminate a report, click the ‘Select’ box next to the report, and then click the ‘Remove’ button below the table. The ‘Remove’ button has the following effect for the different report states:

- For an executing report, ‘Remove’ terminates execution, and removes any generated pages from cache.
- For a pending report, ‘Remove’ deletes the report from the list of pending reports, which prevents the report from executing.
- For a queued report, ‘Delete’ deletes the report from the list of queued reports, which prevents the report from executing.

- For a completed report that remains active in the cache, ‘Remove’ deletes the cached report pages. Subsequent user requests for this report instance will result in a “Session timeout” message.

See Also

Performance Options and Safeguards, to limit the number of simultaneous reports.

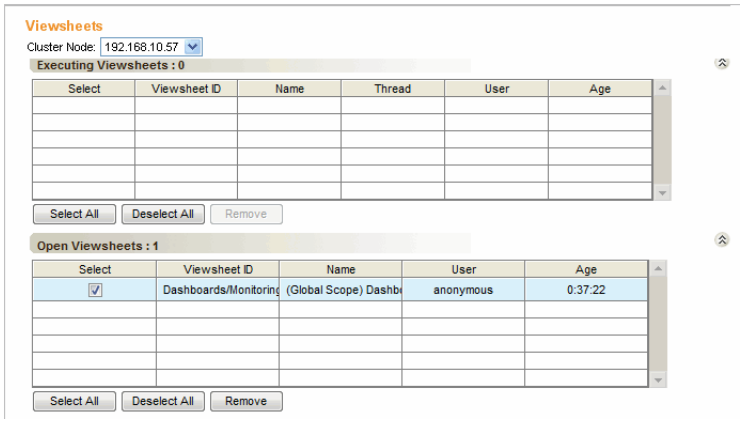
Configuring Remote Monitoring, to accessing data from remote machine.

Configuring Monitoring Resolution, for information on setting the monitoring level.

3.10.3 Viewsheets

The ‘Viewsheets’ page under the ‘Monitoring’ node provides key information about *executing* Viewsheets that are currently being processed by the server, as well as *open* Viewsheets that are active in memory but are not currently being processed.

In a clustered environment, select a machine from ‘Cluster Node’ menu to view the monitoring statistics for that machine.



The page provides the following information:

Table 5. Viewsheet Execution

Viewsheet ID	A unique identifier for the Viewsheet.
Thread	The thread within which the Viewsheet is executing. Click the link to obtain a stack trace.
Name	The name of the Viewsheet.
User	The user who opened the Viewsheet.
Age	The time elapsed since the Viewsheet was opened.
Accessed	The time elapsed since the last Viewsheet operation.

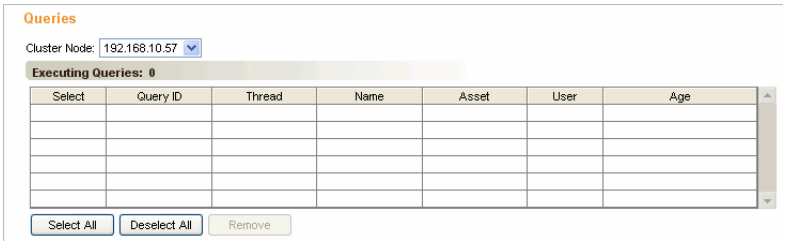
To terminate a Viewsheet, click the ‘Select’ box next to the Viewsheet, and then click the ‘Remove’ button below the table. The ‘Remove’ button has the following effect for the different Viewsheet states:

- For an executing Viewsheet, ‘Remove’ terminates execution, and removes any cached data.
- For an open Viewsheet, ‘Remove’ deletes any cached data.

See Also
Setting Permissions, to control access to Viewsheets.
Configuring Remote Monitoring, to accessing data from remote machine.
Configuring Monitoring Resolution, for information on setting the monitoring level.

3.10.4 Queries

The ‘Queries’ page under the ‘Monitoring’ node provides key information about queries currently executing on the server. In a clustered environment, select a machine from ‘Cluster Node’ menu to view the monitoring statistics for that machine.



The page provides the following information:

Table 6. Query Execution

Query ID	A unique identifier.
Data Objects	The number of dataset rows cached in memory or disk.
Thread	The thread within which the query is executing. Click the link to obtain a stack trace.
Name	The name of the query, or of the worksheet/data model executing the query.
Asset	The name of the report or Viewsheet from which the query was executed.
User	The user who executed the query.
Rows	The number or query rows that have been processed. (Requires ‘Monitoring Level’ set to ‘Medium’ on the ‘Settings’ page.)
Age	The length of time that the query has been executing.

To terminate an executing query, click the corresponding ‘Select’ box to select the query, and then click the ‘Remove’ button.

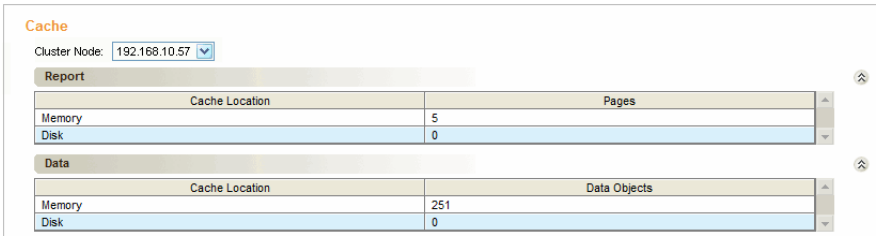
See Also

Configuring Remote Monitoring, to accessing data from remote machine.

Configuring Monitoring Resolution, for information on setting the monitoring level.

3.10.5 Cache

The ‘Cache’ page under the ‘Monitoring’ node provides key information for both the report cache and the data cache. In a clustered environment, select a machine from ‘Cluster Node’ menu to view the monitoring statistics for that machine.



The page provides the following information:

Table 7. Dataset and Report Caching

Pages	The number of report pages cached in memory or disk.
Data Objects	The number of dataset rows cached in memory or disk.
Hits	The number of cache *hits*. A hit occurs when a query is performed against the cache and a match is found. (Requires ‘Monitoring Level’ set to ‘High’ on the ‘Settings’ page.)
Misses	The number of cache *misses*. A miss occurs when a query is performed against the cache and no match is found. (Requires ‘Monitoring Level’ set to ‘High’ on the ‘Settings’ page.)
Swapped	The number of bytes swapped from disk to memory, or from memory to disk. (Requires ‘Monitoring Level’ set to ‘Medium’ on the ‘Settings’ page.)

See Also

Configuring Monitoring Resolution, for information on setting the monitoring level.

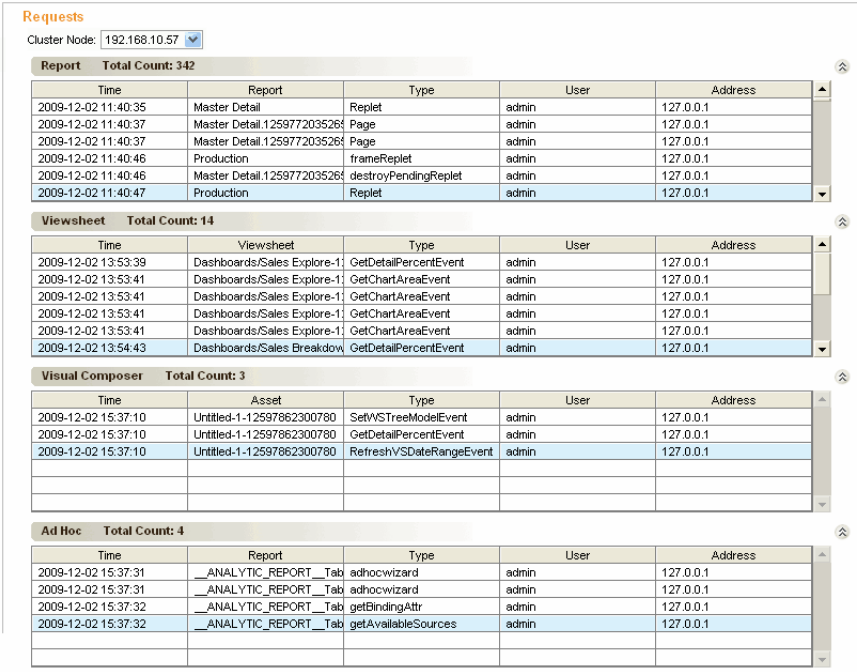
Configuring Remote Monitoring, to accessing data from remote machine.

Performance Options and Safeguards, for information on configuring caching.

3.10.6 Requests

Requires 'Monitoring Level' set to 'High' on the 'Settings' page.

The 'Requests' page under the 'Monitoring' node provides key information about *requests* of various kinds that are being sent to the server: Report, Viewsheet, Visual Composer. In a clustered environment, select a machine from 'Cluster Node' menu to view the monitoring statistics for that machine.



The page provides the following information:

Table 8. Request

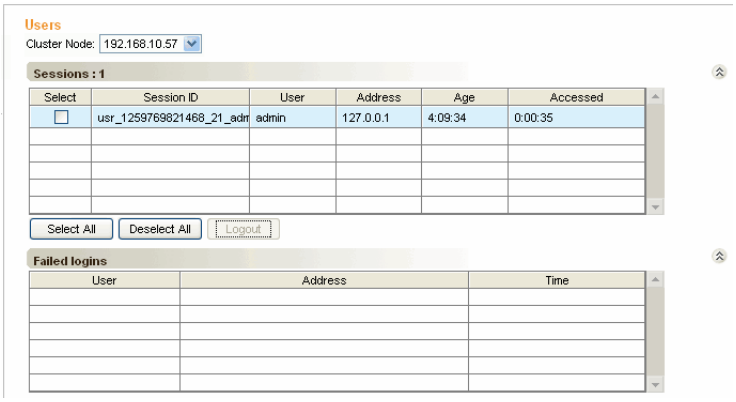
Time	The time at which the request was received.
Report	The identifier for the report being processed.
Viewsheet	The identifier for the Viewsheet being processed.
Asset	The identifier for the asset being processed in Visual Composer.
Type	The type of request received.
User	The user who submitted the request.
Address	The IP address from which the request was received.

See Also
Performance Options and Safeguards, to place limits on server activity.
Configuring Remote Monitoring, to accessing data from remote machine.
Configuring Monitoring Resolution, for information on setting the monitoring level.

3.10.7 Users

Requires ‘Monitoring Level’ set to ‘High’ on the ‘Settings’ page.

The ‘Users’ page under the ‘Monitoring’ node provides key information about user logins, and allows you to terminate a given user session. In a clustered environment, select a machine from ‘Cluster Node’ menu to view the monitoring statistics for that machine.



The page provides the following information:

Table 9. User

Session ID	The unique session identifier.
User	The user who attempted to log in.
Address	The IP address from which the login was attempted.
Age	The elapsed time since start of session.
Accessed	The elapsed time since the most recent user activity.
Time	The time at which the user login was attempted.

To terminate a user session, click the ‘Select’ box next to the desired user, and then click the ‘Logout’ button.

See Also

Users, Roles, and Groups, for information on defining users and login credentials.

Configuring Remote Monitoring, to accessing data from remote machine.

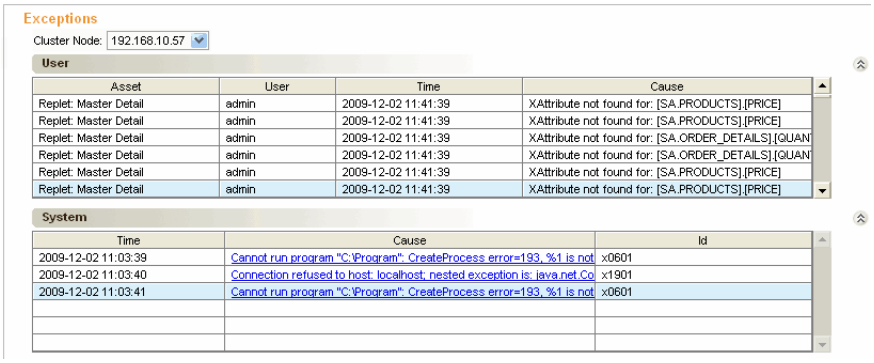
Configuring Monitoring Resolution, for information on setting the monitoring level.

3.10.8 Exceptions

Requires ‘Monitoring Level’ set to ‘High’ on the ‘Settings’ page.

The ‘Exceptions’ page under the ‘Monitoring’ node provides key information about asset exceptions and system exceptions. An *asset exception* is one that causes a report or Viewsheet to fail. A *system exception* is one that causes a critical system failure, such as out-of-memory.

In a clustered environment, select a machine from ‘Cluster Node’ menu to view the monitoring statistics for that machine.



The page provides the following information:

Table 10. Exceptions

Asset	The report or Viewsheet that generated the exception.
User	The user on whose session the exception occurred.
Time	The time at which the exception occurred.
Cause	A description of the exception. Click the link to view a stack trace.
ID	The system exception ID.

See Also

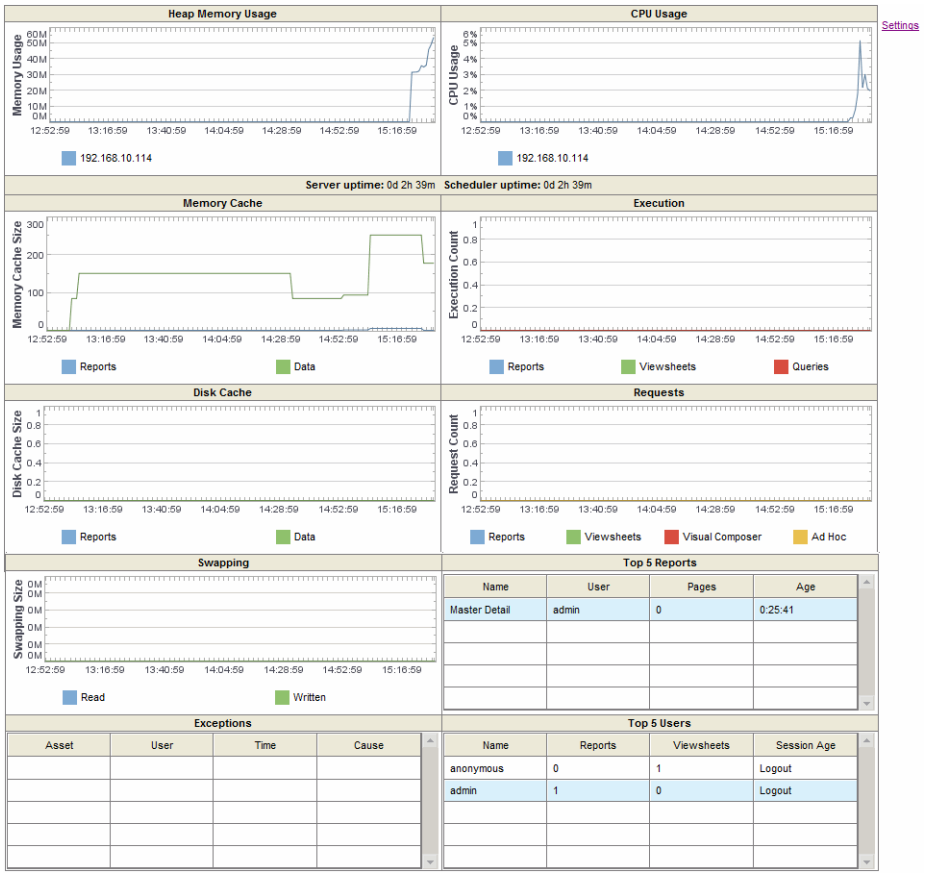
Configuring Remote Monitoring, to accessing data from remote machine.

Configuring Monitoring Resolution, for information on setting the monitoring level.

3.10.9 Summary

The ‘Summary’ page under the ‘Monitoring’ node provides a set of monitoring charts that help you track server performance. In a clustered environment, select a machine from ‘Cluster Node’ menu to view the monitoring statistics for that machine.

Click on a chart to view the detailed information provided by the corresponding page under the ‘Monitoring’ node.



The following information is available. The ‘Level’ column indicates the monitoring level required for the feature.

Table 11. Monitoring Summary

SUMMARY	NOTES	LEVEL
Heap memory usage	for all cluster nodes	None
CPU usage	for all cluster nodes	None
Server uptime		‘Low’ or higher.
Scheduler uptime		‘Low’ or higher.
Memory cache usage	for reports, data	‘Medium’ or higher.
Execution	for reports, Viewsheets, queries	‘Low’ or higher
Disk cache usage	for reports, data	‘Medium’ or higher
Requests	for reports, Viewsheets, Visual Composer	‘High’
Swapping	“written” to disk cache, and “read” into memory cache	‘Medium’ or higher

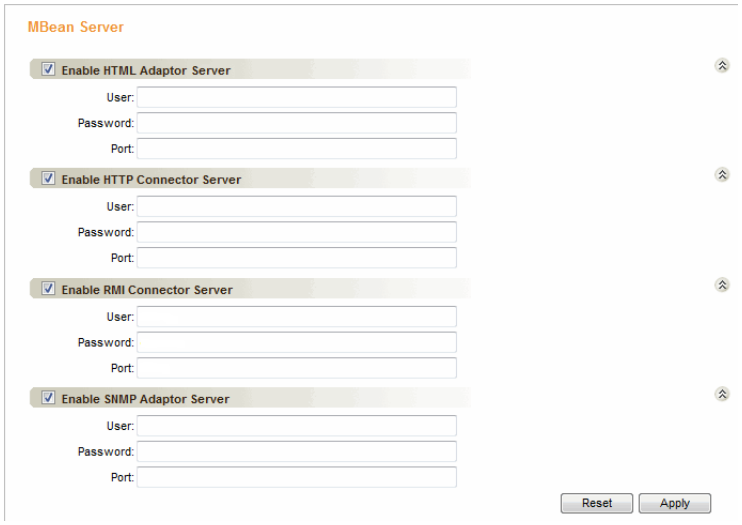
Top 5 reports	by number of pages	‘Low’ or higher; ‘Medium’ to view ‘Pages’ column.
Exceptions	five most recent failures	‘High’
Top 5 users	by count of active reports and Viewsheets	‘Low’ or higher

See Also

Configuring Remote Monitoring, to accessing data from remote machine.
Configuring Monitoring Resolution, for information on setting the monitoring level.

3.10.10 Configuring Remote Monitoring

Because Style Intelligence monitoring features are based on Java’s *managed bean* (MBean) technology, you can access monitoring information from a remote computer. To configure remote access, open the ‘MBean’ page under ‘Server’ node.



This page allows you to configure several methods for accessing the monitoring data. The following sections provide additional information.

See Also

Monitoring Node, to view monitoring dashboards within Enterprise Manager.

HTML Adaptor

The HTML adaptor enables you to view the MBean monitoring information using a web browser on a remote machine. To enable remote HTML monitoring, follow the steps below:

1. Configure the MBean server:

a. On the ‘MBean Server’ page in Enterprise Manager, select ‘Enable HTML Adaptor’.

```
Note: User/password is unique to this module and is not
authenticated by the security provider.
```

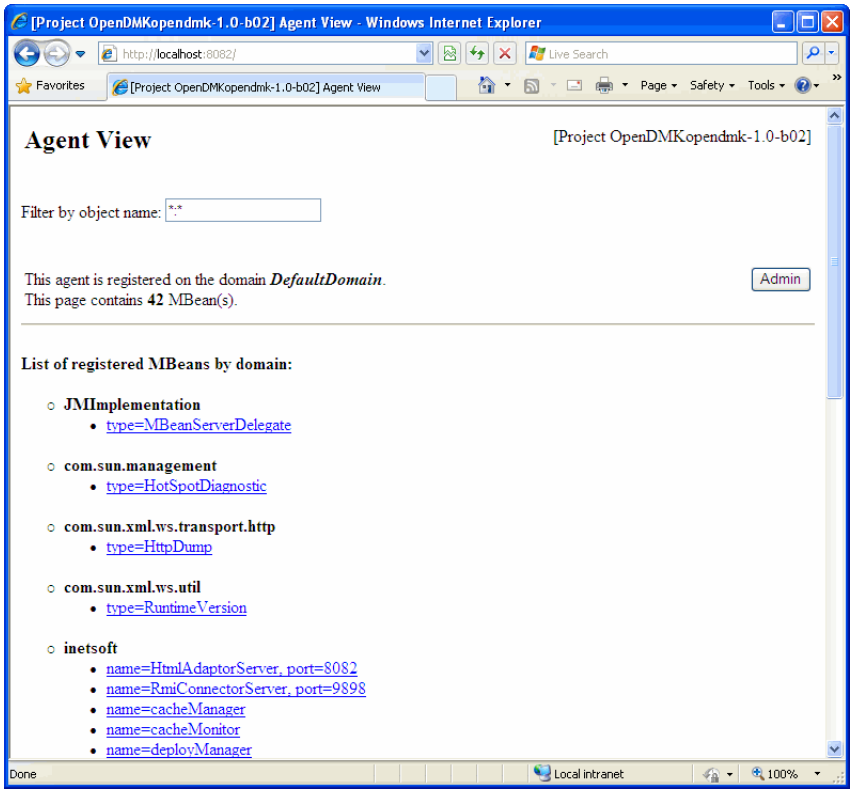
b. Enter a username and password for the connection (default: admin/admin).

c. Enter an arbitrary port number that is not already in use (default:8082).

2. On the remote machine, open a browser and enter the URL shown on the ‘MBean Server’ page: `http://{hostname}:port`.

3. When prompted, enter the previously specified credentials.

This will provide access to the monitoring properties provided by the MBean.



HTTP Connector Server

The HTTP adaptor enables you to view the MBean monitoring information from a remote client that supports the HTTP protocol. To enable remote HTTP monitoring, configure the MBean server as follows:

1. On the ‘MBean Server’ page in Enterprise Manager, select ‘Enable HTTP Connector Server’.

```
Note: User/password is unique to this module and is not
authenticated by the security provider.
```

2. Enter a username and password for the connection (default: admin/admin).

3. Enter an arbitrary port number that is not already in use (default:8081).

RMI Connector Server

The RMI connector server enables you to view the MBean monitoring information using a Java application on a remote machine. As an illustration, the instructions below explain how to access the MBean using JConsole on a remote computer.

For example, to access the MBean using JConsole on a remote machine, follow these steps:

1. Configure the MBean server:

    a. On the ‘MBean Server’ page in Enterprise Manager, select ‘Enable RMI Connector Server’.

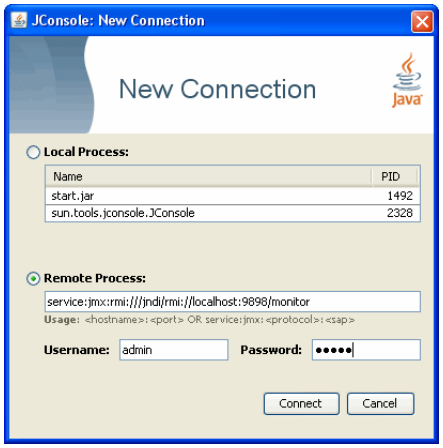
```
Note: User/password is unique to this module and is not
authenticated by the security provider.
```

    b. Enter a username and password for the connection (default: admin/admin).

    c. Enter an arbitrary port number that is not already in use (default:9898).

2. On the remote machine, start JConsole: Navigate to the `%JAVA_HOME%/bin/jconsole` directory, and type “`jconsole`”. This opens the JConsole connection dialog box.

3. In the JConsole connection dialog box, select the ‘Remote Process’ option.



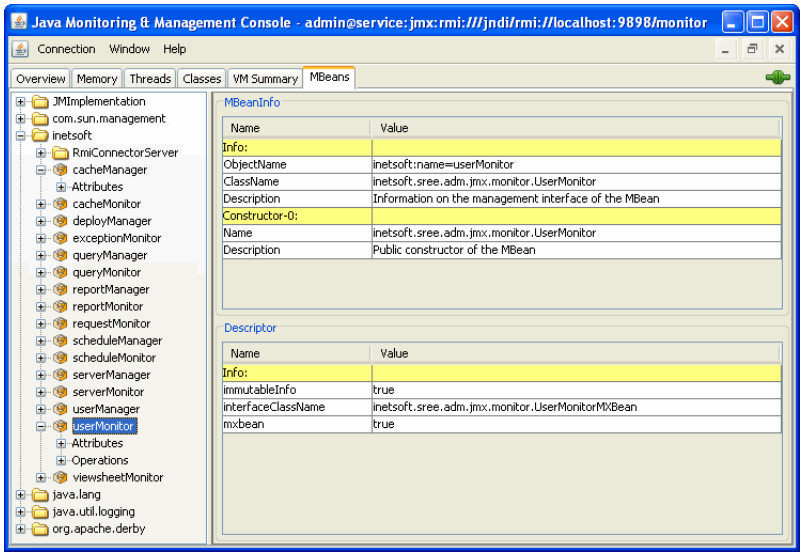
4. In the large text field, enter the following connection string, where `{hostname}` is the server name and `{port}` is the specified RMI port (default: 9898).

   ```
   service:jmx:rmi:///jndi/rmi://{hostname}:{port}/monitor
   ```

5. Enter the previously specified credentials for ‘Username’ and ‘Password’.

6. Click ‘Connect’. This opens the JConsole application.

7. Select the **MBeans** tab, and expand the ‘inetsoft’ node. This provides access to the Style Intelligence monitoring MBean.



SNMP Adaptor Server

The SNMP (Simple Network Management Protocol) adaptor enables you to view the MBean monitoring information from a remote client that

supports the SNMP protocol. To enable remote SNMP monitoring, configure the MBean server as follows:

1. On the ‘MBean Server’ page in Enterprise Manager, select ‘Enable SNMP Adaptor Server’.

```
Note: User/password is unique to this module and is not
authenticated by the security provider.
```

2. Enter a username and password for the connection (default: admin/ admin).

3. Enter an arbitrary port number that is not already in use (default:8085).

4 Repository

The nodes under the 'Repository' node facilitate the configuration of the following properties:

- Storage environment for resources and repository files.
- Properties related to report archiving.
- Audit schedule.
- Persistence of cached files.
- Persistence of queued reports.
- Disk quotas for individual groups and roles.
- Features for sharing assets and monitoring asset dependencies.
- Optimization of Viewsheet data retrieval performance. See *Creating a Materialized View* and *Managing Materialized Views* for more information.

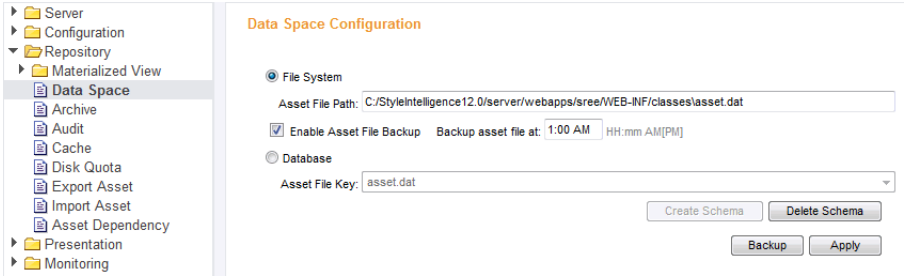
4.1 Data Space

Data space refers to the full collection of all reporting resources (templates, repository files, etc.) stored together in the server repository (SREE Home). This section provides information on how to set up a data space and how to manage the resources in the data space.

4.1.1 Configuring the Data Space

To configure the data space location (i.e., the server repository or "SREE Home") select the 'Repository' > 'Data Space' node on the navigation tree under the **Server** tab. This opens the 'Data Space Configuration' page where the storage type for the data space can be specified. Two storage types are supported: 'File System' and 'Database'. These are explained in the following sections.

Figure 11. Data Space Configuration



File System Data Space

When 'File System' is selected as the data space storage type, the data space is the SREE Home directory (e.g., *WEB-INF/classes*, by default). See *Repository Directory* for information on changing this default location.

If the data space was previously configured for 'Database' storage, you can import the existing data space files into the file system. See *Ant Task for Full Deployment* for more information.

The 'Asset File Path' property specifies the path to the *asset.dat* file containing Data Worksheets and Viewsheets. Enable the 'Asset File Backup' option to create an automatic backup of this asset file at the specified recurring time.

Database Data Space

To use 'Database' as the storage type, ensure that the database is correctly configured in the Enterprise Manager under 'Configuration' > 'Database'. (See *Configuring Database for Audit/Data Space* for more information). The data space can then be configured to use the database as follows:

1. Access the 'Data Space Configuration' page ('Repository' > 'Data Space' in the navigation tree, under the **Server** tab).

2. Select 'Database' as the storage type.

3. Click on the 'Create Schema' button to generate the database schema to be used for the data space, and click 'Apply'.

4. (Optional) If the database contains multiple asset packages (Worksheets/Viewsheets) from a previous import, select the appropriate package in the 'Asset File Path' field.

5. If the data space was previously configured for 'File System' storage, you can import the existing data space files into the database. See *Ant Task for Full Deployment* for more information.

6. Press the 'Apply' button to save all the changes.

To clear the data space when using the database storage type, click 'Delete Schema'. This is useful when you want to re-deploy the application.

When database storage is specified for the data space, newly created reports will be added to the database at the time that they are deployed (from within Style Studio or from the Enterprise Manager). Alternatively, reports can be added to the database by uploading files under the **Data Space** tab.

See Also

Configuring a Repository, to set Style Studio to share a server repository.

4.1.2 Managing Resources in the Data Space

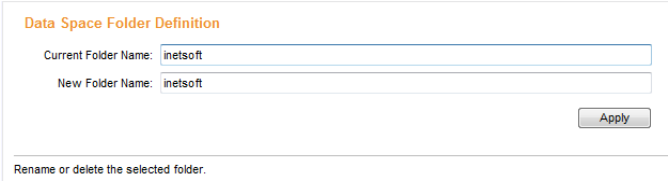
All files and folders in the ‘Data Space’ can be managed through the **Data Space** tab.

Managing Folders

To access a folder’s properties, click the appropriate folder name on the navigation tree. This opens the ‘Data Space Folder Definition’ page.

- To rename the folder, enter a name in the ‘New Folder Name’ field and click ‘Apply’.
- To delete a folder, click the ‘Delete’ link below the navigation tree.
- To add a new file or folder to the selected folder, click the ‘New File’ or ‘New Folder’ link below the navigation tree, respectively.

Figure 12. Data Space Folder Definition

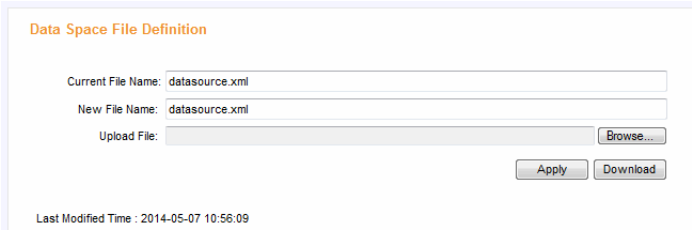


Managing Files

To access a file’s properties, click the appropriate file name in the navigation tree. This opens the ‘Data Space File Definition’ page.

- To rename the file, enter a name in the ‘New File Name’ field and click ‘Apply’.
- To update the contents of the file, click the ‘Browse’ button next to the ‘Upload File’ field. Select a (different) file that contains the updated contents you wish to use, and click ‘Apply’. This overwrites the current contents of the file in the data space with the contents of the selected file.
- To download the file from the data space to the local file system, click the ‘Download’ button.
- To delete the file from the data space, click the ‘Delete’ link below the navigation tree.

Figure 13. Data Space File Definition page



4.1.3 Backing up the Data Space files

You can back-up the resources in the 'Data Space' as a zip archive. Follow the steps below:

1. Under the **Server** tab, select 'Data Space' from the 'Repository' tree node. This opens the 'Data Space Configuration' page.

2. Click on the 'Backup' button. This opens the 'Backup' dialog box.

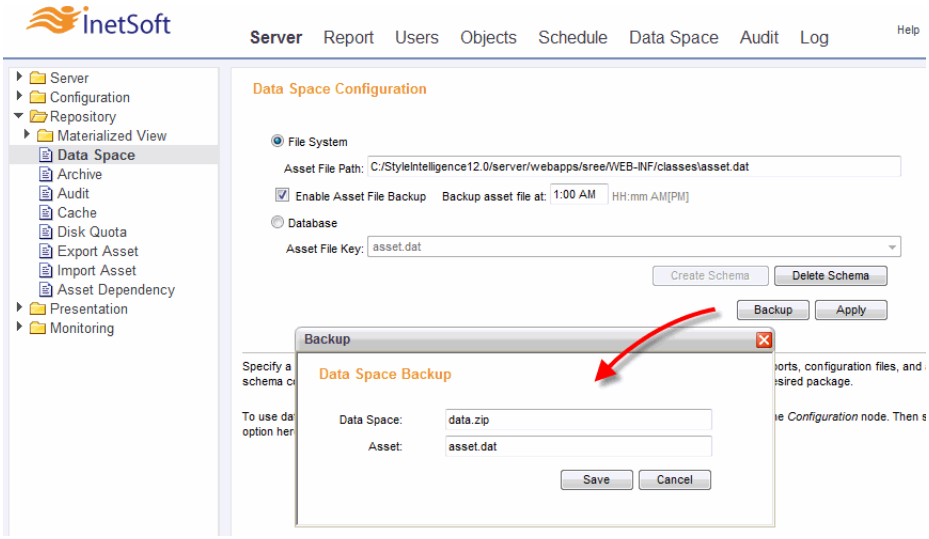
3. In the 'Data Space' field, specify the backup file name, e.g., `data.zip`.

4. In the 'Asset' field, specify a name for the asset file. (The asset file contains Worksheets and Viewsheets.)

`Note: The zip file also contains an asset file (asset.dat, by default). However, this asset file is superseded by the asset file saved under the name you enter in the 'Asset' field. The asset file in the 'Asset' field contains your most recent work.`

5. Click on the 'Save' button. This saves the new zip archive in the SREE Home directory.

Figure 14. Data Space Backup



Note: Dependent assets can be selectively exported by using the 'Repository' > 'Export Asset' node in the **Server** tab. See the *Incremental Deployment of Assets* section for more information.

4.2 Reports, Viewsheets, Worksheets

The report *repository* is the collection of deployed reports (replets) and Viewsheets. You can manage the repository through the **Report** tab in the Enterprise Manager. Reports and Viewsheets can be grouped into folders for organization, and folders can be hierarchically nested. When a folder is deleted, all of the subfolders and related reports will be deleted as well. If a security provider is selected, users (and roles) can be given read and write permission for either folders or individual reports and Viewsheets. Both replet classes and report templates can be registered as replets. The repository can be viewed by users through the User Portal or alternative viewer.

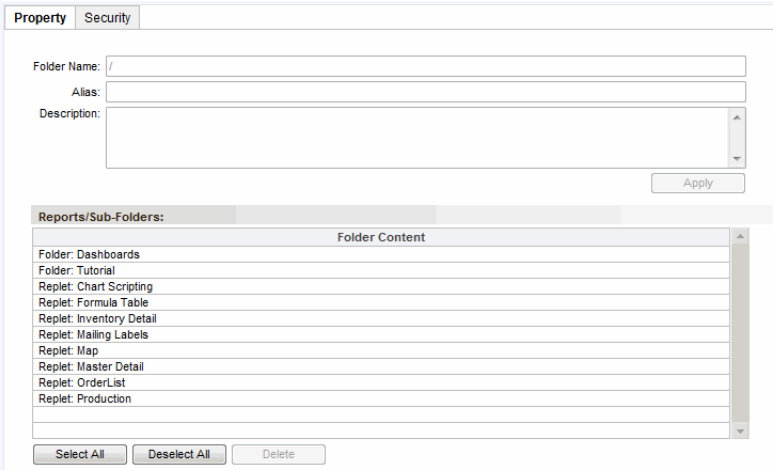
4.2.1 The Repository Page

From the 'Repository' node, under the **Report** tab, you can do the following tasks:

- View a list of folders, reports, snapshots, archived reports, Viewsheets (dashboards), and Data Worksheets
- Add new folders and reports
- Click-and-drag on items in the Repository tree to move them between folders.
- Specify properties for reports, folders, Viewsheets, and Worksheets
- Register new prototype reports

- Delete objects from the Repository tree

Figure 15. Repository Page



4.2.2 Configuring Reports

The definitions of all the registered reports are stored in the repository file named *repository.xml*. Definitions from multiple *repository.xml* files can be edited in the Enterprise Manager if you manually change the `replet.repository.file` property in the *sree.properties* file. Each repository must be separated in the list by a semicolon. All replets registered in these multiple repositories will be available under the 'Repository' node.

To deploy a new report click the 'New Report' link, which appears below the navigation tree. This opens the 'Report Definition' page, which consists of four tabs, **Report**, **Options**, **Parameter**, and **Security**.

Reports in the 'Repository' tree (both in Enterprise Manager and User Portal) are listed in alphabetical order according to the report name specified in the 'Name' field. To reverse the listing, use the 'Sort Repository Tree' option under *Look-and-Feel*. To display reports in an arbitrary order, use the report 'Name' to control the ordering, and use the report 'Alias' to specify the displayed names. For example:

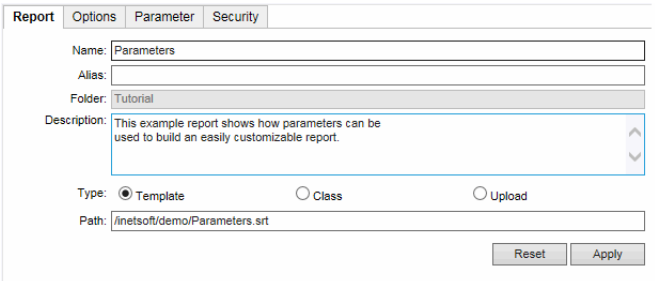
REPORT NAME	REPORT ALIAS
report1	First Report
report2	Middle Report
report3	Last Report

This forces the reports to appear in the desired logical order (First Report, Middle Report, Last Report), which is different than the alphabetical order.

Report Tab

Click the **Report** tab to access the report definition page.

Figure 16. Report Definition Page



The following options are available:

Table 4. Report Definition – Report Options

FIELD	DESCRIPTION
Folder	The folder that contains the report. To change a report's folder, drag the report to a new folder in the Repository tree.
Name	A logical name for the report. Certain characters (e.g., "/") are not permitted in report names.
Alias	The report name displayed to the user in the Portal.
Type	A report can be either a java program that conforms to the report specification or an xml (.srt) template created in the report designer. The Upload option lets you upload a file from your machine/network.
Path	The fully qualified class name of the report, or the template expressed as a java resource.

Options Tab

Click the **Options** tab to access the following options:

Table 5. Report Definition – Options

Priority	Affects thread priority at report generation time. A long-running report can be placed in a lower priority so that it would not block other reports.
Layout Method	Determines how tables are converted to DHTML. The HTML Layout method can be 'Default', 'Auto', 'CSS-P' or 'Table'. 'Default' will use the global layout method; otherwise, the layout method is explicitly set for the report being registered on this page. 'Auto' will cause the layout method to be selected automatically. 'CSS-P' refers to implementing tables as a Cascading Style Sheet, as opposed to the 'Table' layout method which uses the normal HTML tables.
Visible	Specifies whether the report is visible to users.

Bursting	Determines the report's bursting behavior. See *Report Bursting* for additional information.
Pre-generated	Determines whether the report will be pregenerated. If selected, the pregeneration cycle needs to be selected, specifying when the pregeneration should be done. See *Report Pregeneration* for additional information.
Pregeneration cycle	Specifies a pregeneration cycle for reports. Pregeneration cycles can be created/edited under the **Schedule** tab.
Export Formats	Specifies the formats that will be available to the end-user in the Portal for exporting the report. This report-specific setting overrides the setting on the 'Export Menu Options' (see *Report Export Menu Options*.) page of the 'Presentation' > 'Report' node.

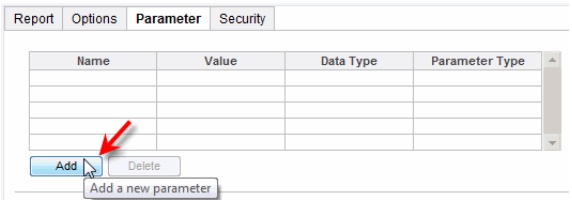
Parameter Tab

Click the **Parameter** tab to manage report initialization and creation parameters.

```
Note: Values that you assign to parameters here will also be used
for report preview in Style Studio (for reports saved as
"repository reports".
```

To automatically pass parameters to the report when it is generated, follow the steps below:

1. Press the 'Add' button below the table on the **Parameter** tab.



   This opens the 'Add Parameter' panel.

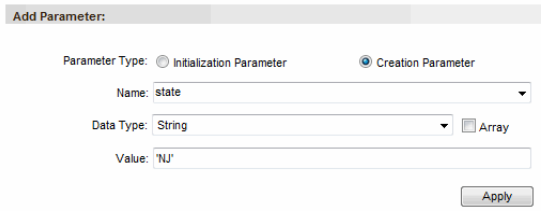
2. Select 'Initialization Parameter' or 'Creation Parameter'.

   Creation parameters are parameters for template-based (.srt) reports. Use these to set default parameter values and bypass the default parameter prompting mechanism (not custom Parameter Sheets). *Initialization parameters* are used to set global properties at initialization for a manually coded Replet class. Some of these are generic and can be used for all report types. (Report creation parameters can also be set from the **Scheduler** tab; see *Creation Parameters*.)

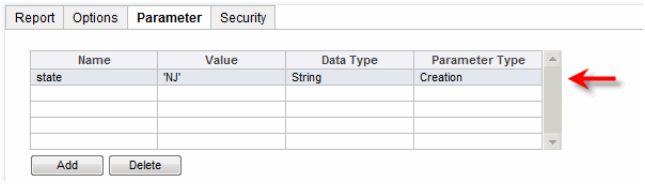
3. In the 'Name' field, select an existing report parameter or manually enter the name of a report parameter.

4. In the ‘Data Type’ field, select the data type of the parameter. Select ‘Array’ if the parameter requires an array of values.

5. In the ‘Value’ field, enter the value to be passed into the parameter. Enter a comma-separated list for an array parameter.



6. Press ‘Apply’. The parameter that you added is now listed in the table.



7. (Optional) To modify the parameter settings, click on the parameter in the table and make the desired edits below the table. To remove the parameter, press the ‘Delete’ button below the table.

To add additional parameters, press the ‘Add’ button below the table and repeat the above steps.

Users can also sort columns with the Report Explorer. See *Sorting a Table* in *Using a Report*.

One popular initialization parameter is the Boolean ‘sortOnHeader’. Set this to ‘true’ to enable users to click on a table column header and sort the rows by that column. (Note: ‘sortOnHeader’ cannot be enabled for Freehand tables, or when data has not been bound to the table using Style Studio.)

Security Tab

Click the **Security** tab to manage report permissions.

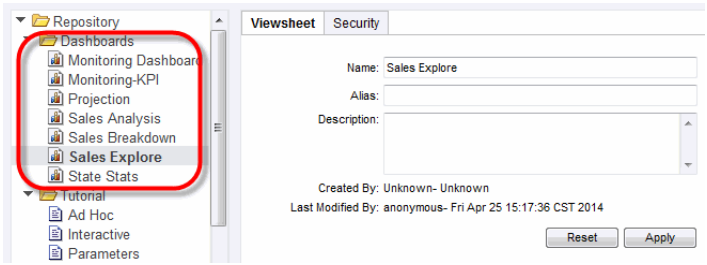
By default, a report is set to inherit security permissions from its parent, which is the folder it resides in. This default behavior can be modified and a report may have its own security information setup, independent of its parent. Once set up, the security permissions for the report override those of the report’s parent. Setting up security for a report is very similar to setting up security for any other component. See the *Setting Permissions* section of the *Security* chapter.

Please note that if a template which uses JavaScript is registered as a report, the package names may also need to be registered in order to be accessible from the scripting environment. Any inetsoft.* packages are automatically registered. Other packages can be specified, separated by commas, in the javascript.java.packages property.

Note that reports can also be deployed from within Style Studio. See *Deploying an Asset* for more information.

4.2.3 Configuring Viewsheets

Viewsheets appear in the Repository under the 'Dashboards' node.



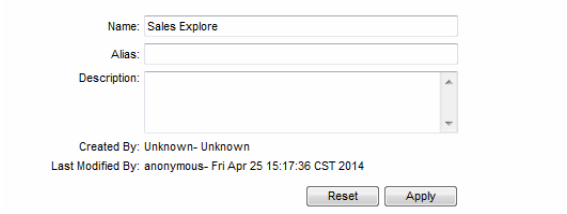
The definitions of all Viewsheets are stored in the repository file named *asset.dat*. To add a Viewsheet to the Repository, use the Visual Composer to save the Viewsheet into a global scope. See *Advanced Dashboard Design* for more details.

To modify settings for a particular Viewsheet, click the desired Viewsheet in the Repository tree. This opens the 'Viewsheet Definition' page, which consists of three tabs: **Viewsheet**, **Materialized View**, and **Security**. The next sections provide additional details about the available settings.

Viewsheet Tab

Click the **Viewsheet** tab to access the Viewsheet definition page.

Figure 17. Viewsheet Definition Page



The following options are available

Table 6. Viewsheet Definition – Viewsheet Options

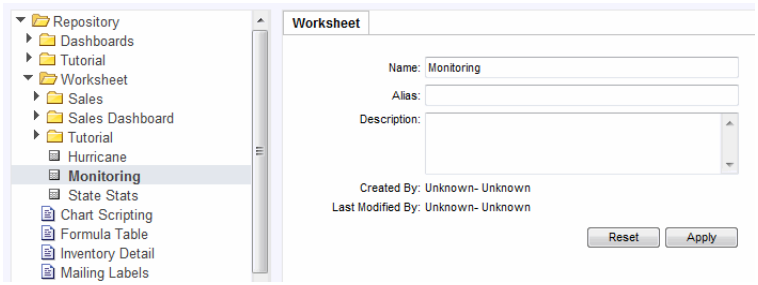
FIELD	DESCRIPTION
Name	The name under which the Viewsheet should be listed in the Repository. Certain characters (e.g., "/") are not permitted in names.
Alias	The Viewsheet name displayed to the user in the Portal.
Description	A brief description to be displayed as a tooltip when the user hovers the mouse over the Viewsheet in the Portal or Visual Composer Repository trees.

Security Tab

Click the **Security** tab to manage Viewsheet permissions. By default, a Viewsheet is set to inherit security permissions from its parent, which is the folder it resides in. This default behavior can be modified and a Viewsheet may have its own security information setup, independent of its parent. Once set up, the security permissions for the Viewsheet override those of the Viewsheet's parent. Setting up security for a Viewsheet is very similar to setting up security for any other component. See the *Setting Permissions* section of the *Security* chapter.

4.2.4 Configuring Data Worksheets

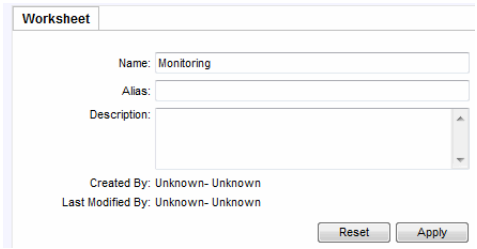
Data Worksheets appear in the Repository under the 'Worksheet' node.



The definitions of all Data Worksheets (other than local Worksheets embedded within individual reports) are stored in the repository file named *asset.dat*. To add a Worksheet to the Repository, use the Visual Composer to save the Worksheet into a global scope or use Style Studio to deploy the Worksheet to the repository. See *Advanced Data Mashup* for more details.

To modify settings for a particular Worksheet, click the desired Worksheet in the Repository tree. This opens the 'Worksheet Definition' page.

Figure 18. Worksheet Definition Page



The following options are available

Table 7. Worksheet Definition – Worksheet Options

FIELD	DESCRIPTION
Name	The name under which the Worksheet should be listed in the Repository. Certain characters (e.g., "/") are not permitted in names.
Alias	The Worksheet name displayed to the user in the Portal.
Description	A brief description to be displayed as a tooltip when the user hovers the mouse over the Worksheet in the Portal or Visual Composer Repository trees.

4.2.5 Report Pregeneration

A pregenerated report is similar to a cached report, and helps improve performance and scalability. From the user's perspective, a pregenerated report behaves like any live report and supports all user interactions. To the reporting engine, a pregenerated report appears as an archived report whose data is generated on a pre-defined cycle and maintained in persistent storage. When the user accesses a pregenerated report, the report loads the pregenerated data instead of executing the query and page generation process.

Note: A VPM cannot be applied to *pre-generated* reports. See *Virtual Private Model Security* in *Advanced Data Modeling* for more details.

To configure a report template as a pregenerated report, enable 'Pre-generated' on the **Options** tab of the report definition page (see *Options Tab*), and specify the cycle that determines when the report is pregenerated.

4.2.6 Creating Folders

To create a new folder, follow the steps below:

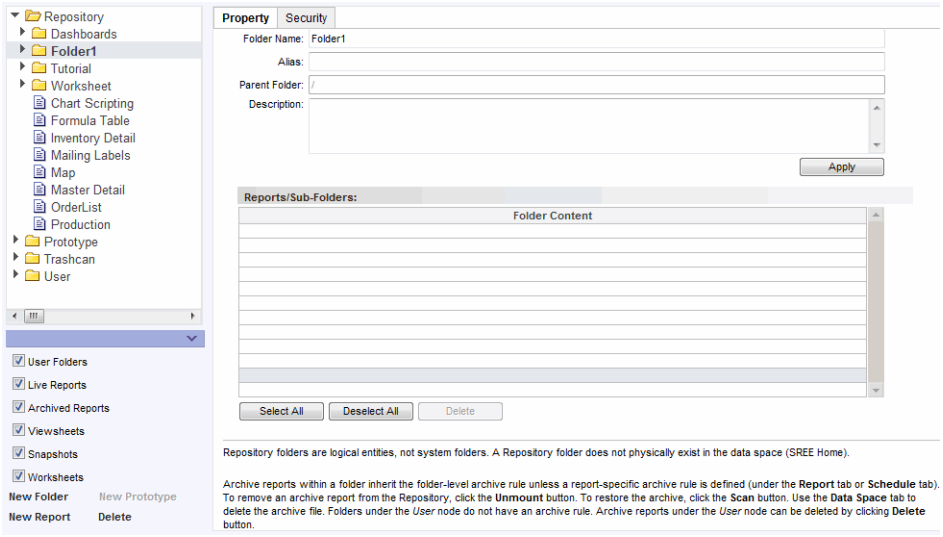
1. In the navigation tree, select a parent folder for the new folder. The parent folder can be the root 'Repository' node or any other folder. (You can change the parent folder later, if needed.)

2. Click the ‘New Folder’ link below the navigation tree. This opens the folder definition page, which has two tabs: **Property** and **Security**.

3. Under the **Property** tab, specify the new ‘Folder Name’ and ‘Description’.

4. (Optional) Set the folder ‘Alias’. The alias is displayed to the user in the Portal in place of the folder’s actual name. If no alias is specified, the folder’s name is used.

5. Set the folder’s permissions using the **Security** tab. See *Setting Permissions* for more information.



The contents of the folder are displayed in the ‘Reports/Sub-folders’ table. Use the ‘Select All’, ‘Deselect All’, and ‘Delete’ buttons as needed to remove folder contents. To add a new report to the folder, drag the report into the folder in the Repository tree. (See *Configuring Reports* for more information.)

4.2.7 Report Bursting

This section covers the second step in implementing report bursting. The first step is documented in the *Advanced Report Design*. To follow along with the following example, refer to the *Report Bursting* chapter in the *Advanced Report Design* and create the sample report ‘Bursting Example’. Report Bursting relies on the Scheduler system. Reports are set to generate when certain conditions are met and then saved with the complete data set. More information on the Scheduler may be found in the *Scheduler* chapter.

Walkthrough

A report is bursted out to end users when they request the report after logging into the system or when reports are emailed to them. Here you will concentrate on the execution of a report with bursting options.

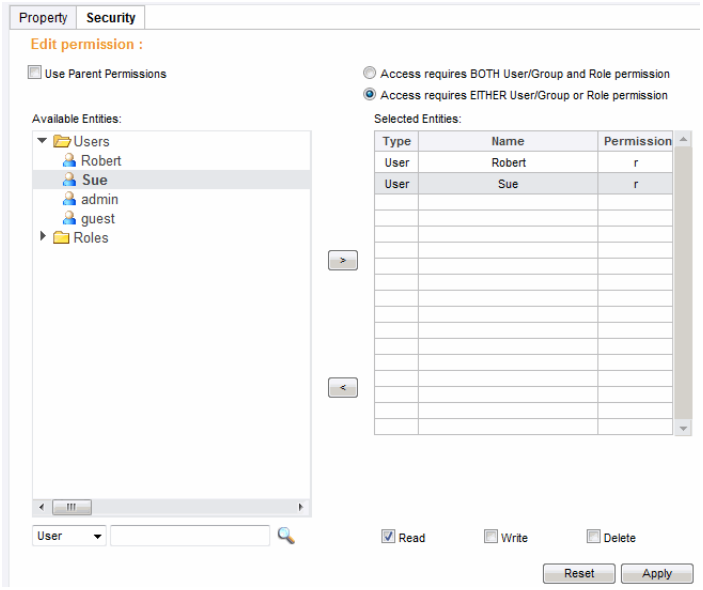
1. Open the Enterprise Manager and enable the 'Default Security Provider'. Complete instructions for this step and the next step may be found within the *Configuring the Default Security Provider* section.

2. Create a user Sue and a user Robert, each with 'abcd1234' as their password. Also, specify both of their email addresses.

3. Under the **Report** tab, click the 'Repository' node and select the **Security** tab.

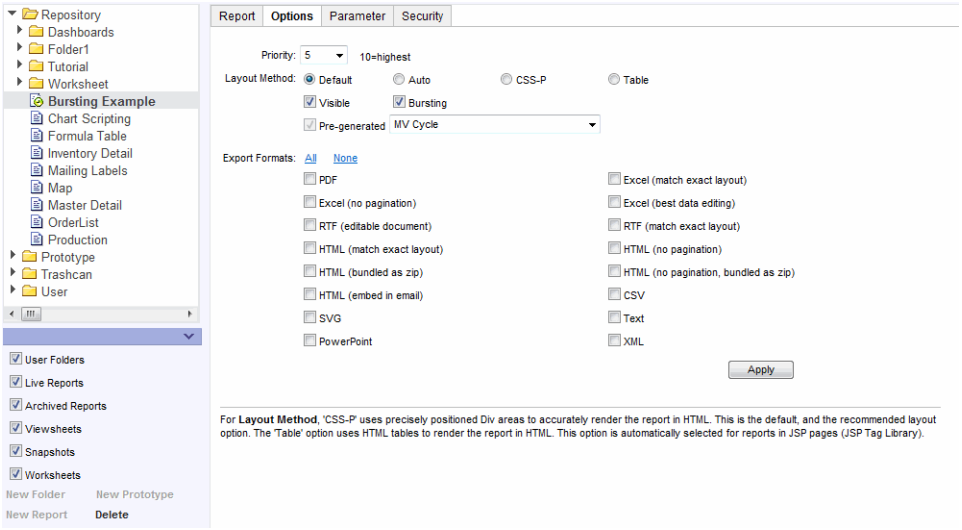
4. Deselect the 'Use Parent Permissions' checkbox.

5. Assign 'read' permissions to both users.



6. Open the report repository listing, select the 'Bursting Example' report you created in the *Report Bursting* chapter of the *Advanced Report Design* example, and then select the **Options** tab.

7. Select the 'Bursting' option to turn on bursting. The pregenerated cycle can be selected from the menu below. For this example, select 'Default'. This refers to the cycle that will be used to generate the report data in the Scheduler. Press the 'Apply' button.



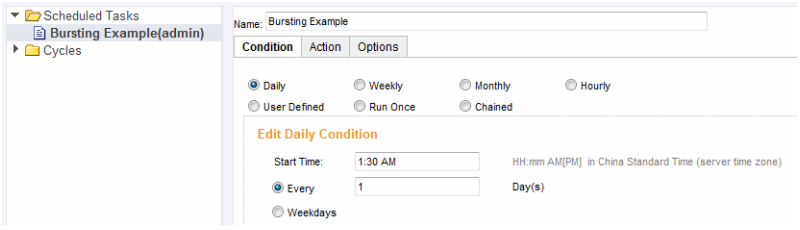
8. Access the ‘Server’ > ‘Scheduler’ page from under the **Server** tab and ensure that the Scheduler is running. If not, click the ‘Start’ button.

9. Go to the **Schedule** tab and access the list of scheduled tasks.

10. Click on the ‘New Task’ link, located below the navigation tree.

11. Rename the task to “Bursting Example” and press the ‘Apply’ button, located next to the text box.

12. For the condition, specify a ‘Daily’ condition. Set a time of 12:00 am for this example and press the ‘Apply’ button. This condition must be met for the full report to be generated. In this example, you will run the report immediately by using the ‘Run Now’ feature in the Scheduler.



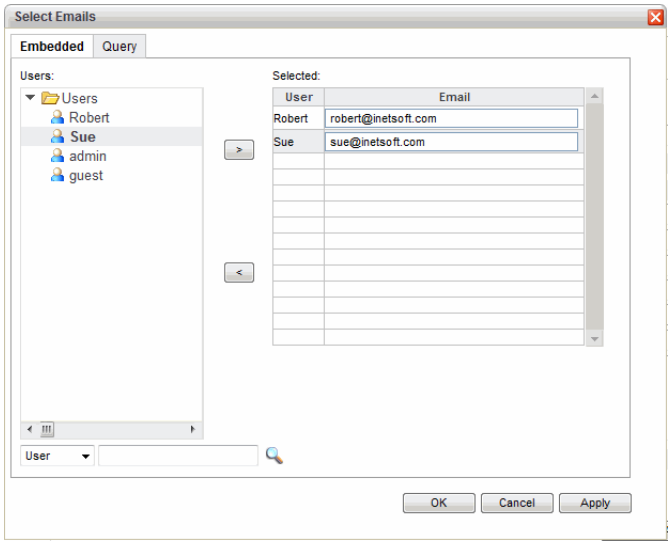
13. Select the **Action** tab and choose ‘Burst’ as the Action Type.

14. In the ‘Report’ drop down list, select ‘Bursting Example.’

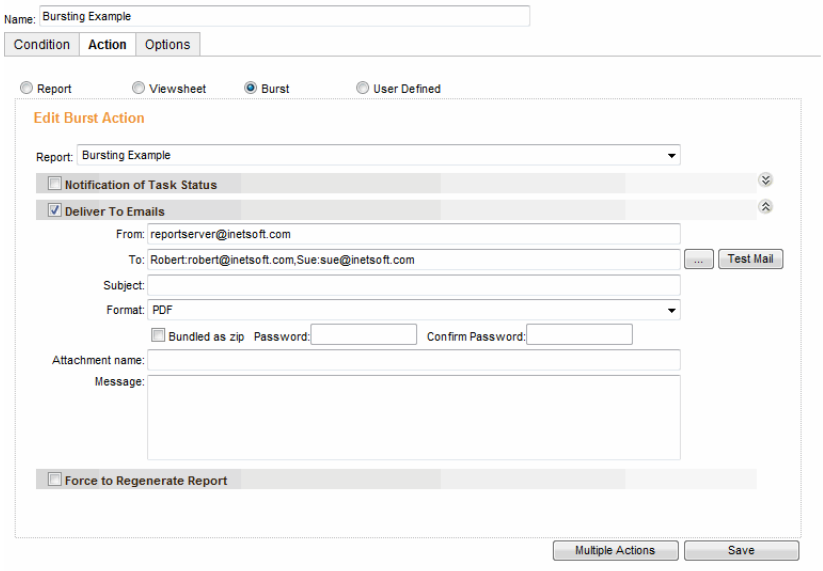
Note: The 'Select Emails' dialog box is available only when the 'Execute As' property (**Options** tab) specifies an administrator.

15. Select the 'Deliver to Emails' option, and click on the '...' button next to the 'To' textbox to. This opens the 'Select Emails' dialog box.

16. Add Robert and Sue as the users to whom the email should be sent. Click on the 'OK' button.



17. Click on the 'Apply' button to save the action definition.



18. From the 'Server Tasks' root node, select the 'Bursting Example' task and press 'Run Now' to immediately execute the task.

The task emails two different versions of the report, each version providing the user with their unique view of the bursted data.

4.3 User Reports

The User Portal provides many ways for users to create new reports or customize existing reports to suit their needs. Once created or modified, the report can be saved in the user's personal folder, named 'My Reports'. Users also have the ability to save archived reports under the 'My Reports' folder.

Managing User Reports

Individual user's reports can be managed under the **Report** (tab) > User folder. This folder contains a sub-folder for each individual user in the system. The appropriate folder node can be expanded to view/manage the list of a particular user's reports. A generic description for all user folders (My Reports) can be entered under the **Property** tab of the 'User' folder.

4.4 Archive

Reports are inherently dynamic. Whenever a user sends a request to generate a report, the engine executes the report and retrieves fresh data from the database, regardless of whether the same report has been generated before. For small to medium sized reports, the impact of generating the report each time is minimal and generally does not cause any performance problem.

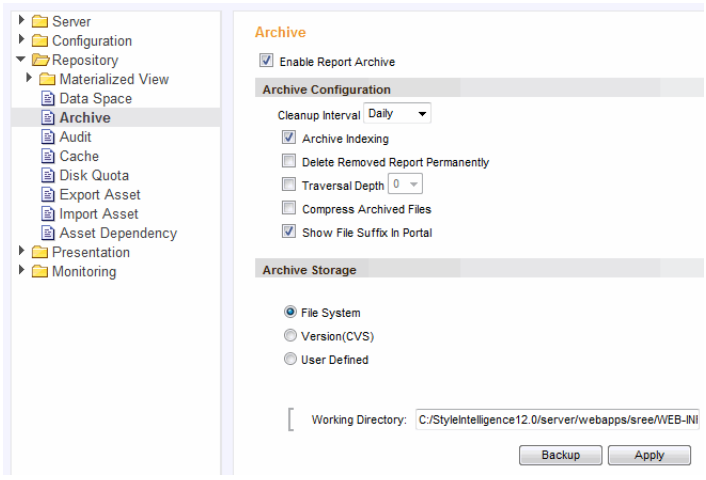
However, if an application has many very large reports, dynamically generating the reports on demand may not be a feasible solution. To avoid repeatedly generating the reports, a report archive for the application can be set up to save and reuse generated reports.

4.4.1 The Archive Page

You can configure report archiving from the 'Archive' page, under the 'Repository' node of the **Server** tab. Report archives support the following features:

- Saving and retrieving reports
- Report versioning
- Report access permissions
- Archive rules for cleaning up saved reports
- Different report formats: Generated Report, PDF, XML, Excel, PowerPoint, RTF, HTML, CSV, and SVG
- Searching the archived reports for a particular string
- Backing up an archive
- Compressing archived files

Figure 19. The Archive Page



4.4.2 Archive Types

This software supports three types of report archives: File system-based archive, CVS-based archive, and user-defined archive.

- **File System**: The 'File System' option saves and manages archived reports on the server file system. This type of archive only allows one version per report. The report files are managed internally by the report archive. Therefore, the archive does not depend on any external software. See *File System Archive* for more information.
- **Version (CVS)**: The 'Version (CVS)' option selects a versioned archive based on CVS. All reports are saved and managed through a CVS system. Each report can be saved in multiple versions. In order for the archive to work, the 'CVS Bin' field must specify an absolute path to the CVS executable file (cvs.exe). See *CVS Archive* for more information.
- **User Defined**: The 'User Defined' option may be used as an alternative to the default archives. A common user-defined archive would, for instance, save the archived report into a user database. The user-defined archive class needs to implement the ReportArchive or VersionedArchive interface. See *User-Defined Archive* for more information.

See Also

Archived Reports, for information on accessing archived reports.

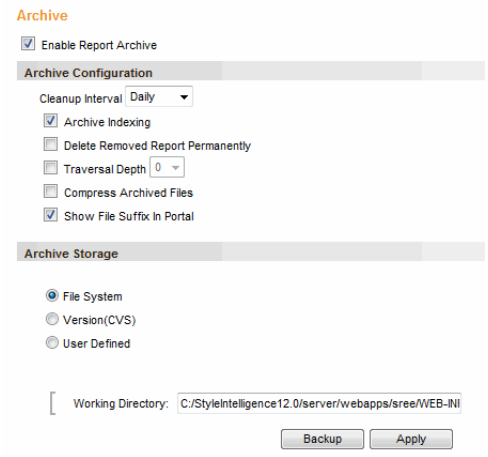
Administering Archived Reports, for information on managing archived reports.

4.4.3 File System Archive

The file system archive uses a working directory to hold the saved reports and administrative files. The directory does not need to be empty. However, it is strongly recommended that a specifically dedicated directory be used as the working directory, so the files are not accidentally removed or modified.

Setting up a file system archive requires the specification of the working directory location on the server:

1. Start the Enterprise Manager.
2. Select the **Server** tab.
3. Expand the 'Repository' node and select 'Archive' on the Navigation tree.
4. Select 'File System' as the type of Archive Storage.
5. Specify the working directory, which should reside within the repository directory.



6. Click on the 'Apply' button to configure the archive.
7. Specify desired configuration parameters, listed under 'Archive Configuration'.

The cleanup interval controls how often the archive is cleaned up. The cleanup process is controlled by the archive rules specified in the **Schedule** tab for the archived folders and reports. See the *Administering Archived Reports*section for information about tagging reports for deletion. Daily cleanup is performed at 1am every morning. Weekly cleanup is performed

at 1am every Sunday. Monthly cleanup is performed at 1am on the first day of the month.

4.4.4 CVS Archive

Note: Reports archived using versions of Style Report prior to 5.1 may only work with PDF export.

'CVS' archiving uses the UNIX standard Concurrent Versioning System (CVS) to store the saved reports. 'CVS' is not included with this product. It is normally distributed on UNIX systems, but can also be downloaded from the Internet.

To use the 'CVS' archive, the CVS executable (cvs.exe) must be accessible on the report server.

1. Start Enterprise Manager. Select the **Server** tab.

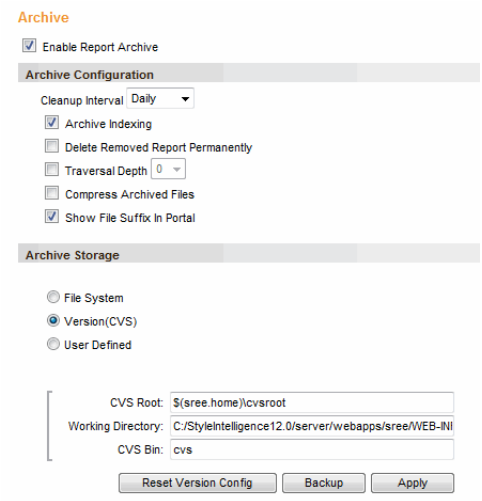
2. Expand the 'Repository' node and select 'Archive' on the Navigation tree.

3. Select 'CVS' as the type of Archive Storage.

4. Specify 'CVS Root' directory. The CVS root can be a remote directory previously configured using CVS commands (e.g., *pserver:user@machine:/{path}*), or it can be a new archive directory. If the specified root directory does not already exist, it is automatically created.

5. Specify the 'Working Directory' that holds the temporary files. This should be a directory that is dedicated to the archive, and it should be located within SREE Home.

6. Specify 'CVS Bin', the full path (using forward slashes) to the CVS executable file. If not specified, the system's default path will be searched.



After changing any CVS setting, always click ‘Apply’ followed by ‘Reset Version Config’, and then restart the server.

7. Click ‘Apply’, and then click ‘Reset Version Config’ to configure the archive.

8. Restart the server.

4.4.5 User-Defined Archive

You can customize archiving in three ways:

- Use standard data archiving mechanisms provided by Style Intelligence, but customize the storage of the archive files. Choose ‘User Defined’ under the ‘Archive Storage’ heading and then choose ‘Storage Class’ from the radio button below. Enter the fully qualified name of your custom storage class. The custom storage class must implement one of the following interfaces:

  ```
  inetsoft.sree.store.DataStorage
  inetsoft.sree.store.VersionedStorage
  ```

- Customize the data archiving mechanism but use the standard storage mechanisms provided by InetSoft. Choose ‘User Defined’ under the ‘Archive Storage’ heading and then choose ‘Archive Class’ from the radio button below. Enter the fully qualified name of your custom archive class. The custom storage class must extend one of the following classes, classes depending on the type of standard storage you would like to use:

  ```
  inetsoft.sree.store.impl.DefaultReportArchive
  inetsoft.sree.store.DefaultVersionedArchive
  ```

- Customize both the data archiving mechanism and the storage of archived files. Choose ‘User Defined’ under the ‘Archive Storage’ heading and then choose ‘Archive Class’ from the radio button below. Enter the fully qualified name of your custom archive class. The custom storage class must extend the `inetsoft.sree.store.impl.DefaultReportArchive` class. Override the `getStorageClass()` method to return the fully qualified class name of your custom storage class. The custom storage class must implement one of the following interfaces:

  ```
  inetsoft.sree.store.DataStorage
  inetsoft.sree.store.VersionedStorage
  ```

Refer to the product API documentation for more details on the custom Archive Classes and Interfaces.

4.4.6 Archived Reports

This section explains how to use archived reports.

Archiving a Report

A report can be saved into the archive through a scheduled task or by a user through their Portal page, using the ‘Save as archive’ option in the ‘Save As’ dialog box. If a file system archive is being used, archiving a report with the same name as an existing archived report will cause the original archived report to be overwritten by the new archived report. If a ‘CVS’ archive is being used, saving a report to an existing report adds a new version to the archive, rather than replacing the previous version. The versions are stored until removed explicitly by the user, or according to the version archive rule.

Opening an Archived Report

If a user has at least read permission in an archived report, the archived report shows up on the repository tree on the user’s Portal page. A user can then click on the archived report name to view the report.

4.4.7 Administering Archived Reports

Administrators can manage the reports and the trashcan through the Enterprise Manager.

Removing and Restoring Archived Reports

The administrator can select any report from the tree and delete it, and is not restricted by the access permission of the report. Once a folder or report is deleted, it is moved to the trashcan.

End users can delete a report from the repository tree if they have delete permission on the report. When a report is deleted, it is placed in a trashcan, and can be recovered by an administrator.

The trashcan can be accessed by clicking on the ‘Trashcan’ node on the navigation tree under the **Report** tab. To restore a folder or report, select the folder or report and click ‘Undelete’. The folder or report is recovered from the trashcan and placed back in the archive. A folder or report can be permanently removed from the archive by deleting it from the ‘Trashcan’. After the folder or report is deleted from the ‘Trashcan’, it cannot be restored to the archive.

Similarly, archived reports can be removed from the repository without the file being deleted. This is called ‘Unmount’, and can be done on the detail page of the archived report, or that of its parent folder. At the folder level there is also an option to ‘Scan’ which will look at the physical folder contents and recreate the repository entries. This could also be used to add arbitrary documents to the repository.

Archive Report Options

The following options are available for archived report on the 'Archive' page.

- **Delete Removed Report Permanently**: The archive has the option 'Delete Removed Report Permanently'. If this option is chosen, then when a report is deleted, it is not placed in the trashcan but is permanently deleted from the system.
- **Archive Indexing**: The 'Archive Indexing' option is by default set to 'true'. When this option has been enabled, a report saved in the archive will be indexed for search capabilities. When this flag is set to 'false' the archive search will be disabled.

```
Note: An individual archived report is created to correspond to
each traversed hyperlink, so processing time for deeper traversals
may be substantial.
```

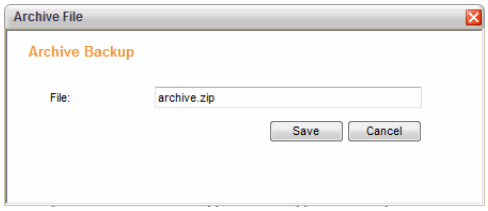
- **Traversal Depth**: The 'Traversal Depth' option allows you to set the number of levels of reports that can be followed starting from the hyperlinks within the archived report. This option only applies to the 'Generated Report' output format. A 'Traversal Depth' of 0 indicates that the reports targeted by hyperlinks in the archived report will not themselves be archived, and the hyperlinks will therefore not be active. A 'Traversal Depth' of 1 indicates that one level of hyperlinked reports will also be archived, which then allows the user to follow hyperlinks in the original archived report down to a second level of archived reports. However, the hyperlinks (if any) in that second level of archived reports will not themselves be active, unless an even higher traversal depth is specified.
- **Backup Archive**: Backup the archive to a zip file. See *Backing-up the Archive* for instructions.
- **Compress Archived Files**: The files saved in the archive can be compressed using this option. Only the files that are not already compressed are compressed if this option is turned on. Compressed files can be accessed normally from the viewer with the file name of the uncompressed file. However, compressed files are saved in the underlying storage as *filename.zip*, for example *report.csv.zip*. The zip file contains a single entry – the archived file, e.g., *report.csv* in the previous example.
- **Show File Suffix in Viewer**: Setting this property to true results in the file extensions being included when the archived reports are listed on the repository tree on the Portal page.

Backing-up the Archive

To backup an archive into a zip file, follow the steps below:

1. Select ‘Repository’ > ‘Archive’ on the Navigation tree under the **Server** tab.

2. Click on the ‘Backup’ button. This opens the ‘Archive File’ dialog.



3. Specify the backup file name and click on the ‘Save’ button.

Note that if the backup file name is specified without a directory, the file is created in the sree.home directory.

4.4.8 Viewsheet Management

Viewsheets can be accessed and managed under the **Report** tab. See *Configuring Viewsheets* for more information.

4.5 Auditing

The audit tool tracks report and data access, as well as asset dependencies. When the audit function is enabled, all report and database access is logged in the audit database. Please note that for auditing to be turned on, a database has to be configured in the Enterprise Manager, and a security provider must be specified. Refer to the *Configuring Database for Audit/ Data Space* section for information on how to configure a database, and refer to *Specifying a Security Provider* for information on the available security options.

4.5.1 Enabling Auditing

Auditing can be turned on and off using the ‘Enable Audit’ checkbox on the ‘Repository’ > ‘Audit’ page under the **Server** tab. To turn on auditing:

1. Go to the ‘Repository’ > ‘Audit’ page under the **Server** tab.

2. Select ‘Enable Audit’.

3. Entering a value in the ‘Store audit records for’ field to specify the number of days that audit records should be stored in the database. By default, the duration is 365 days.

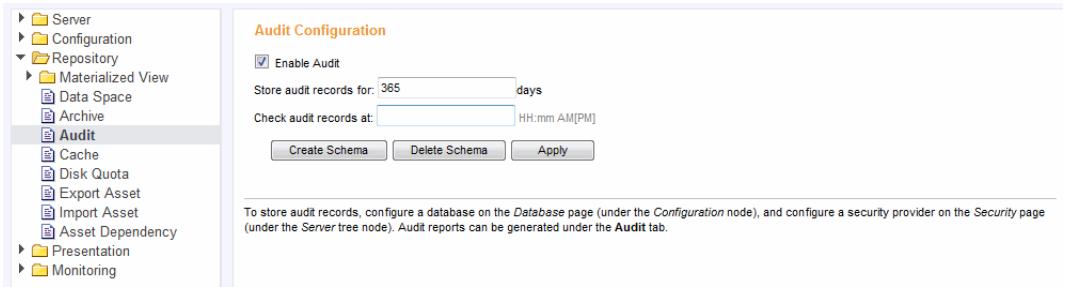
4. Enter a value in the ‘Check audit records at’ field to specify the time of day at which obsolete audit records should be deleted.

5. Click on the ‘Create Schema’ button.

6. Click on the ‘Apply’ button.

All audit logs can be cleared from the database at any time by clicking on the ‘Delete Schema’ button.

Figure 20. Audit Configuration



4.5.2 Viewing Audit Logs

The audit logs can be viewed from the **Audit** tab. The type of logged information under each Audit node is shown in the table below.

Figure 21. Audit Page

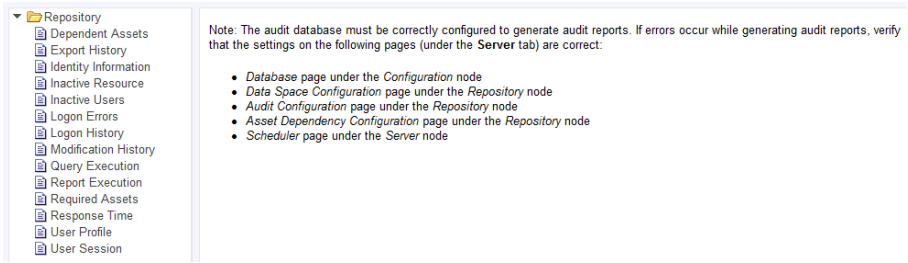


Table 8. Type of information under each Audit Log node

NODE NAME	TYPE OF INFORMATION
Dependent Assets	Report of assets that depend on a selected asset.
Export History	Report of reports and dashboards that have been exported by users.
Identity Information	Report of modifications to user identity information.
Inactive Resource	Report of resources not used over a selected period.
Inactive Users	Report of users who have not logged in over a selected period.
Logon Errors	Report of login errors over a selected period.
Logon History	Report of session logins and logouts over a selected period.
Modification History	Report of assets modified over a selected period.

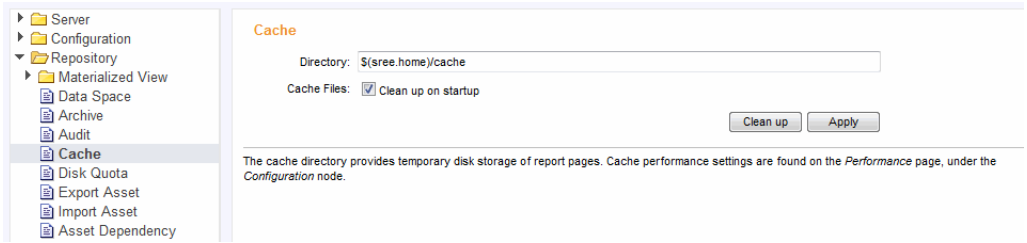
NODE NAME	TYPE OF INFORMATION
Query Execution	Report of queries, data models, and Data Worksheets executed over a selected period.
Report Execution	Report of reports executed over a selected period.
Required Assets	Report of assets that are required by a selected asset.
Response Time	Report of report generation times.
User Profile	Report of users defined in system. (User creation dates are not available for users added prior to enabling auditing.)
User Session	Report of user session times.

4.6 Cache

The report engine possesses an internal page cache. When the user opens a report or Viewsheet, the report engine generates page objects for the report or Viewsheet, and stores these objects in the internal cache.

You can configure this cache from the 'Repository' > 'Cache' page under the **Server** tab. The 'Directory' field specifies the location of the cache directory (`replet.cache.directory`), which by default is the '*temp*' directory in SREE Home.

Figure 22. Report Cache



By default, all temporary files are removed when the server starts up. In cases where multiple applications are using the same cache directory and automatic removal is not desired, you can disable this behavior by deselecting 'Clean up on startup'.

To set the size of the cache, as well as a variety of other performance settings related to caching, see *Tailoring Server Performance*.

4.7 Disk Quota

Disk quotas can be specified for individual groups and roles in order to control space on the file system or in the database. The space occupied by the following components counts toward a user's disk quota:

- Archived reports

- Generated reports in the ‘My Reports’ folder.

A particular user’s disk quota is either:

- The maximum of all the maximum disk quotas set for the groups and roles to which that user belongs, or
- The value specified for the ‘Default Max Space’, if disk quotas are not individually specified for any of the groups or roles to which that user belongs.

Setting the Disk Quota

Disk quotas can be set from the ‘Repository’ > ‘Disk Quota’ page under the **Server** tab. By default, the disk quota is disabled. To set a disk quota, select ‘Quota Enabled’. The default disk quota for all the users in the system can be specified using the ‘Default Max Space’ property. A value of -1 indicates no limit.

It is possible to set a quota for individual groups and roles. This can be done in the following steps:

1. Click on the ‘Add’ button.
2. Select the desired role or group from the appropriate drop down list.
3. Specify the ‘Default Max Space’ in megabytes.
4. Click on the ‘Apply’ button.

4.8 Incremental Deployment of Assets

An *asset* is any reporting unit, which could be a bean, meta-template, dashboard, report, etc., as well as assets within worksheets or Viewsheets. The options described below allow for assets to be selectively exported from the data space and imported into a new environment, thus facilitating incremental deployment. An asset dependency audit can also be performed.

4.8.1 Exporting Assets

The ‘Repository’ > ‘Export Asset’ node under the **Server** tab opens the ‘Export Assets’ page, which allows the developer to combine an arbitrary set of assets into an archive (JAR file). The assets in this archive can then be imported into a different environment (e.g., different server).

For instructions on how to export a set of assets, see *Exporting Assets*. Assets in the following categories can be exported this way:

- Data: Data Models, Queries, Worksheets, etc.
- Global Dashboard: Administrator-defined dashboards.
- User Dashboard: User-defined dashboards.
- Global Repository: Administrator-deployed reports and Viewsheets (which include user-placed bookmarks)
- User Repository: User-deployed reports (‘My Reports’ folders).
- Library: Reusable components; beans, scripts, table styles, etc.

See Also
Ant Task for Incremental Deployment, to programmatically control deployment.

4.8.2 Importing Assets

You cannot import a user-scope asset into an environment which has security disabled, or which has no definition for the particular user.

Assets can be selectively imported into the data space by using the ‘Repository’ > ‘Import Asset’ node under the **Server** tab, which opens the ‘Import Asset’ page. This allows an administrator to incrementally deploy assets to an existing environment without overwriting all the assets in that environment. The assets to import must reside in an archive file generated by the ‘Export Asset’ feature described in *Exporting Assets*.

For instructions on how to import a set of assets from an archive file, see *Importing Assets*.

See Also
Ant Task for Incremental Deployment, to programmatically control deployment.

4.8.3 Ant Task for Incremental Deployment

You can perform incremental import and export of assets programmatically via a custom ANT task. This requires ANT 1.6 or later, and JDK 1.6 or later.

The task can run in two modes:

- Remote import/export to and from a live server deployment via RMI. (See *RMI Connector Server* to set up the RMI service.)
- Local import/export to and from a local file system (Sree Home).

The following sections provide additional information.

Ant Task Parameters

The task uses two classes, *ImportAssetTask* and *ExportAssetTask*, and requires the following parameters:

- `classname` – API class name, based on the desired operation:

```
inetsoft.sree.adm.ant.ExportAssetsTask
inetsoft.sree.adm.ant.ImportAssetsTask
```

- `file` – the absolute file path to which assets should be exported, or from which assets should be imported.
- `replace` – Import flag to overwrite existing assets with the same name.

Remote Import/Export Only

The following parameters pertain to RMI-based asset deployment.

- `host` – the machine or IP address of the RMI server.
- `port` – the RMI port of the RMI Connector Server.
- `username/password` – User/Password of the RMI Connector Server.

Local Import/Export Only

The following parameters pertain to local asset deployment.

- `sreehome` – the absolute path to the sree home folder.

Including and Excluding Assets

The assets to be included or excluded in the import/export are specified using specific patterns within the `<include>` and `<exclude>` tags.

```
Pattern — /scope/asset type/folder/subfolder/asset name
```

For example,

```
Global Scope — /global/worksheet/Sales/YTD Sales
User Scope — /user/{user name}/viewsheet/Lead Analysis
Asset Wild Card — /global/viewsheet/Mgmt Dashboard/Sales/*
Folder Wild Card — /global/report/Mgmt Dashboard/**/*
```

For example, if you want to include all viewsheets from the ‘Sales’ folder except the ‘YTD Sales’ viewsheet.

```
<include name="/global/viewsheet/Sales/*"/>
<exclude name="/global/viewsheet/Sales/YTD Sales"/>
```

Sample *build.xml* file for Remote Import/Export:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<project name="AdminTask" basedir="." default="all">
<property name="testDir" location="${basedir}"/>

<target name="ExportAssetRemote" description="Export From">
<taskdef name="exportassets"
classname="inetsoft.sree.adm.ant.ExportAssetsTask"/>
```

```
<exportassets host="192.168.5.89" port="1098"
username="{RMI Server user}" password="{password}"
file="${testDir}\exports\xyz1.jar">
<include name="/global/viewsheet/Dashboards/*"/>
<exclude name="/global/viewsheet/Dashboards/Projection"/>
</exportassets>
</target>

<target name="ImportAssetRemote" description="Import To">
<taskdef name="importassets"
classname="inetsoft.sree.adm.ant.ImportAssetsTask"/>
<importassets host="192.168.5.89" port="1098"
username="{RMI Server user}" password="{password}"
file="${testDir}\imports\xyz2.jar" replace="false">
</importassets>
</target>
</project>
```

Sample *build.xml* for file for local Import/Export:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<project name="AdminTask" basedir="." default="all">
<property name="testDir" location="${basedir}"/>
<property name="sreeHome"
location="C:\StyleIntelligence\server\webapps\sree\WEB-
INF\classes"/>

<target name="ExportAssetLocal" description="Export From">
<taskdef name="exportassets"
classname="inetsoft.sree.adm.ant.ExportAssetsTask"/>
<exportassets username="{EM User}" password="{password}"
sreehome="${sreeHome}" file="${testDir}\exports\abc1.jar">
<include name="/global/viewsheet/**/*"/>
</exportassets>
</target>

<target name="ImportAssetLocal" description="Import To">
<taskdef name="importassets"
classname="inetsoft.sree.adm.ant.ImportAssetsTask"/>
<importassets username="{EM User}" password="{password}"
sreehome="${sreeHome}" file="${testDir}\imports\abc2.jar"
replace="false">
</importassets>
</target>
</project>
```

Running the Ant Task

To execute the Ant task, follow the steps below:

1. Make sure the product JAR files are on the classpath.
2. Change to the directory containing the `build.xml` file.
3. Enter the following command:

```
ant {target name}
e.g., ant ExportAssetLocal
```

4.8.4 Ant Task for Full Deployment

To move the full contents of the repository from a file system repository to a database repository, use the `ImportFSToDB` Ant task. To move the full contents of the repository from a database repository to a file system repository, use the `ImportDBToFS` Ant task.

Below is a sample *build.xml* file for full deployment:

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="DataspaceImport" basedir="." default="test">

  <path id="project.classpath">
    <path path="C:/StyleIntelligence/server/webapps/sree/
WEB-INF/lib/bisuite.jar"/>
    <path path="C:/StyleIntelligence/server/webapps/sree/
WEB-INF/lib/etools.jar"/>
    <path path="C:/StyleIntelligence/server/webapps/sree/
WEB-INF/lib/derby.jar"/>
    <path path="C:/StyleIntelligence/server/webapps/sree/
WEB-INF/lib/derbyclient.jar"/>
    <path path="C:/StyleIntelligence/server/webapps/sree/
WEB-INF/lib/derbynet.jar"/>
  </path>
  <property name="project.classpath"
refid="project.classpath"/>
  <typedef resource="inetsoft/sree/adm/ant/antlib.xml"
classpath="${project.classpath}" />

  <!-- Import from File System to Database -->
  <target name="ImportFSToDB">
    <importfstodb home="C:/StyleIntelligence/server/
webapps/sree/WEB-INF/classes" forcetooverwrite="false"/>
  </target>

  <!-- Import from Database to File System -->
  <target name="ImportDBToFS">
    <importdbtofs home="C:/StyleIntelligence/server/
webapps/sree/WEB-INF/classes" forcetooverwrite="false"/>
  </target>

</project>
```

The '`home`' directory should contain the *build.xml* file and the database configuration file *dbProp.properties*. To execute the Ant task, follow the steps below:

1. Make sure the product JAR files and database driver JAR files are on the classpath.

2. In a command window, change the working directory to the location containing the *build.xml* file.

3. Enter the following command:

```
ant {target name}
e.g., ant ImportFSToDB
```

If `forcetooverwrite` is set to `false`, you will be prompted at the command line to confirm each overwrite operation. If `forcetooverwrite` is set to `true`, you will not be prompted to confirm overwrites.

4.8.5 Auditing Asset Dependencies

You can produce an audit report for asset dependencies. Asset dependency information describes, for each asset, the other assets which it itself requires, and the assets which require it. The audited assets include every reporting unit, not only worksheets and Viewsheets.

The schedule on which the dependency report is generated can be configured from the 'Repository' > 'Asset Dependency' node under the **Server** tab, which opens the 'Asset Dependency Configuration' page. A database must first be specified in the 'Configuration' > 'Database' age under the **Server** tab.

To configure a regular asset dependency audit, follow these steps:

1. Check the 'Enable Check Asset Dependency' box.

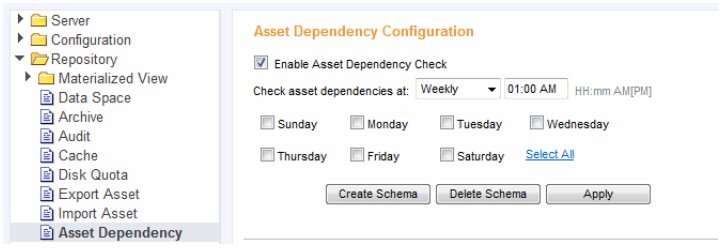
2. Select 'Daily' or 'Weekly' in the 'Check asset dependencies at' field. For the 'Daily' option, specify an execution time. For the 'Weekly' option, specify an execution time and select a set of days on which the task should execute.

3. Press the 'Create Schema' button, and press 'Apply.'

   A schema needs to created in cases where the database has been changed from the default database (under 'Configuration' > 'Database' in the **Server** tab).

4. To delete the schema, click the 'Delete Schema' button, and press 'Apply.'

Figure 23. The Asset Dependency Configuration page.



The asset dependency audit can also be run on demand from the 'Scheduled Tasks' node under the **Server** tab. Select the `__check assets dependency__` task from the task list, and press the 'Run Now' button.

The results of the asset dependency audit can be viewed under the **Audit** tab. Select either the 'Dependent Assets' or 'Required assets' reports to obtain the desired view into the audit database.

4.9 Materialized Views

Materialized views are available only for Viewsheets.

A *materialized view* is a caching strategy that pre-aggregates and stores the data required by a Viewsheet so that the Viewsheet can respond more quickly at runtime. When a materialized view exists for a Viewsheet, the Viewsheet attempts to query the cached materialized view rather than the database. This can significantly reduce the run time for certain queries.

Viewsheets that possess a materialized view will rely on this materialized view to supply their data. Viewsheets that do not possess a materialized view will either query the database directly or automatically trigger the creation of a materialized view. Use the 'Enable MV On-demand' option in *Materialized View Options* to enable the latter behavior.

To improve performance, you can distribute a materialized view across multiple machines. See *Distributing Materialized Views (Data Grid)* for further details.

4.9.1 Creating a Materialized View

To create the materialized view for a Viewsheet, follow the steps below:

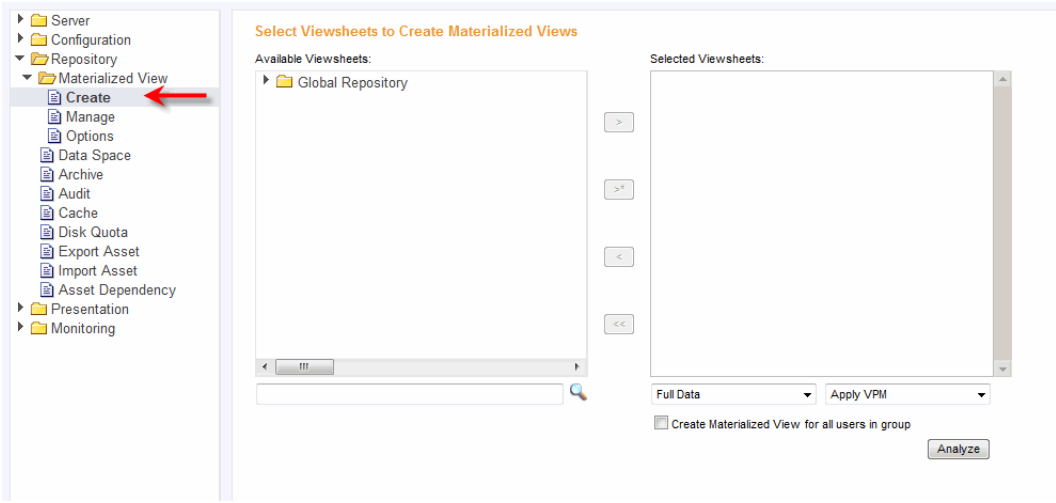
1. Log into Enterprise Manager.

   See *Distributing Materialized Views (Data Grid)* for more information about data grid configurations.

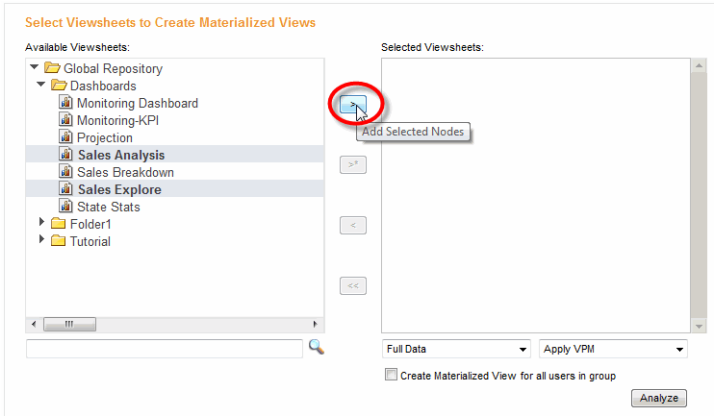
   If you are using a data grid, you must log into Enterprise Manager on the Data Server machine. (You cannot create a materialized view by using Enterprise Manager on the Report Server machine, unless that machine also hosts the Data Server.) For example, if *Machine1* contains the Report Server, *Machine2* contains the Data Server, and *Machine3* through *MachineN* contain the Data Nodes, you must log into Enterprise Manager on *Machine2* to create materialized views:

   ```
   http://Machine2:8080/sree/EnterpriseManager
   ```

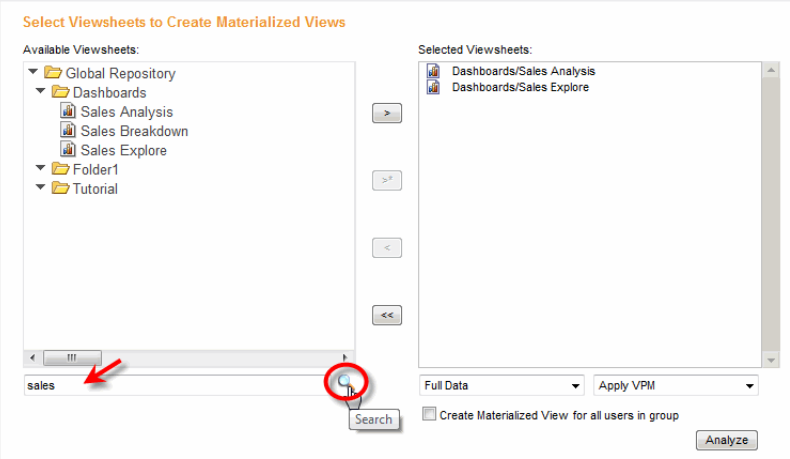
2. Expand the 'Repository' > 'Materialized View' node under the **Server** tab, and select 'Create'.



3. Select one or more Viewsheets or folders from the 'Available Viewsheets' panel, and press 'Add Selected Nodes'. This adds the selected items to the 'Selected Viewsheets' panel.



To filter the list of available Viewsheets based on keyword, enter a desired search term in the search field below the 'Available Viewsheets' list and press the 'Search' button.



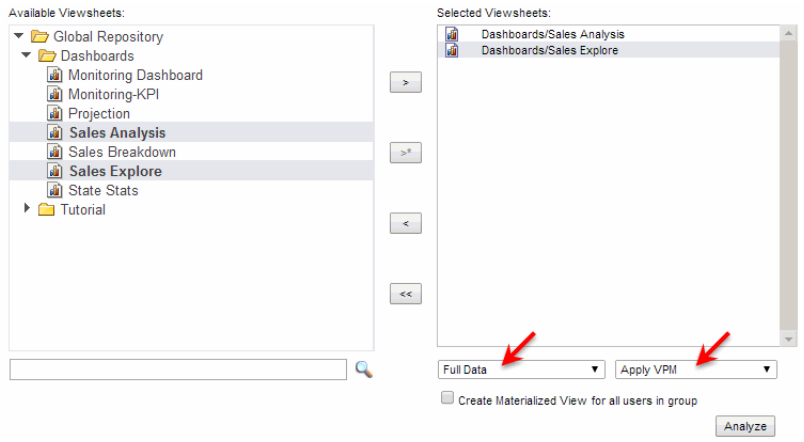
4. From the menu below the ‘Selected Viewsheets’ panel, choose either ‘Full Data’ or ‘Minimum Data’.

   ‘Full Data’ materializes the entire Worksheet Data Block associated with a materializable Viewsheet component. ‘Minimum Data’ materializes only those columns of the Data Block that are directly supplying data to the Viewsheet component.

   The ‘Minimum Data’ option conserves disk space, and is recommended when you do not expect the Viewsheet to undergo further editing after deployment. In cases when the Viewsheet will undergo further development (for example, end-user chart binding modifications), use the ‘Full Data’ option to ensure that all data is available for use.

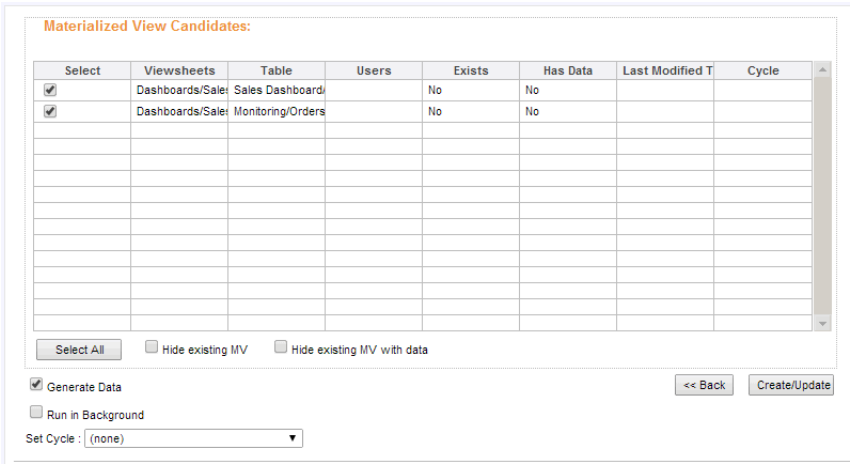
See *Virtual Private Model Security* in *Advanced Data Modeling* for details.

5. (Optional) If you are using VPMs, select ‘Apply VPM’ from the menu below the ‘Selected Viewsheets’ panel if you want to apply existing VPMs when creating the materialized view. If your VPM filters data based on particular users, enable ‘Create Materialized View for all users in group’. This allows a custom materialized view to be generated for each user, if needed. (See *Data Security with Materialized Views* below for more details.) To keep a materialized view free from the effects of any existing VPMs, select ‘Bypass VPM’.

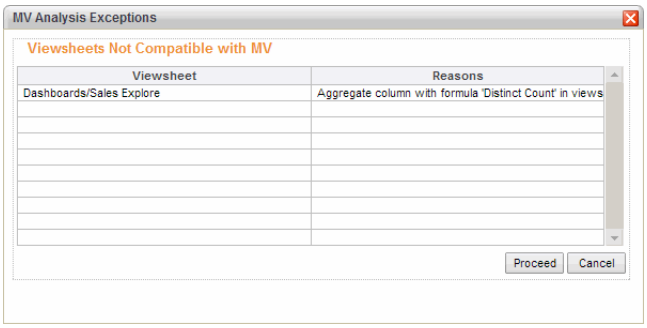


Only Viewsheets containing aggregated data can be considered candidates for materialized views.

6. Press the 'Analyze' button. This opens the 'Materialized View Candidates' page containing a list of Viewsheets that can be materialized.



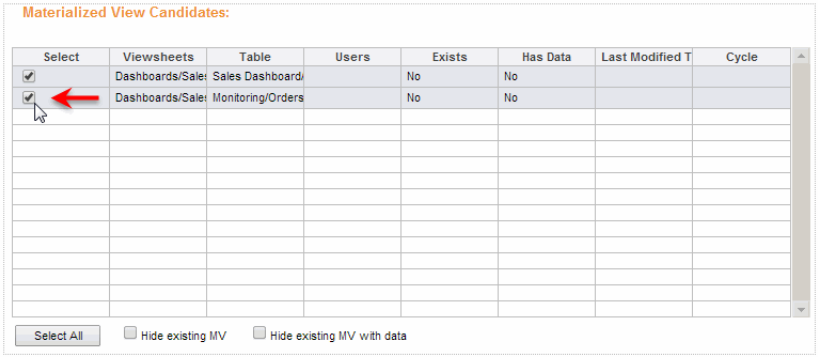
If there are design aspects of a Viewsheet or Data Worksheet that prevent the generation of a materialized view, the 'MV Analysis Exceptions' dialog box is shown.



Press ‘Proceed’ to continue with the materialization process for any materializable Viewsheets, or press ‘Cancel’ to abort the materialization process. See *Transformations and Faults* for information about design features that may interfere with materialization.

7. (Optional) Select ‘Hide existing MV’ to remove a Viewsheet from the list if the Viewsheet already possesses a materialized view (with or without data). Select ‘Hide existing MV with data’ to remove a Viewsheet from the list if the Viewsheet already possesses a materialized view that has been populated with data.

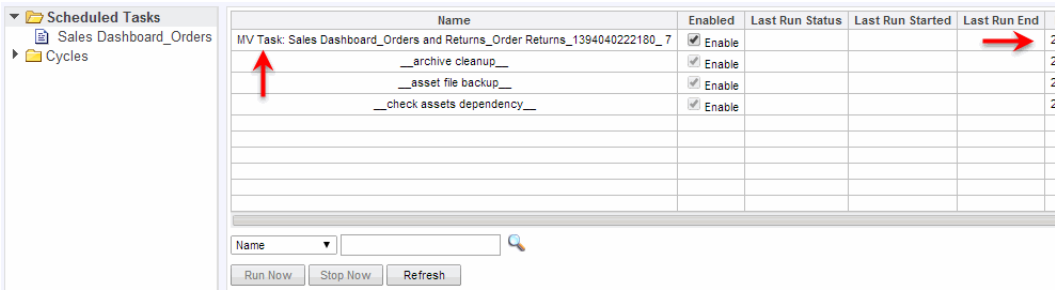
8. Check the boxes in the ‘Select’ column to select the Viewsheets for which you want to create a materialized view. To select all Viewsheets in the table, press the ‘Select All’ button below the table.



Note: To schedule a materialized view in a data grid environment, you must first set the mv.repository.servlet property in *sree.properties* to the complete URL of the Data Server. This allows the Scheduler to communicate the request to the web application.

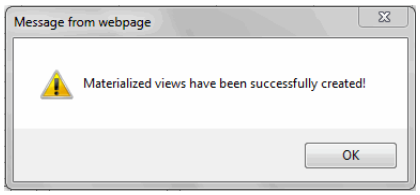
9. In the ‘Select Cycle’ column of the table, choose the cycle on which the materialized view should be regenerated. (See the *Scheduler Cycle* section for more information about defining cycles.)

10. (Optional) Select ‘Generate Data’ if you want to immediately perform the full materialization process for the selected Viewsheets upon pressing the ‘Create/Update’ button. (This may take a significant amount of time, during which the materialized view is constructed and data files are saved into the *WEB-INF/classes/bs* directory.) If you do not want to wait for materialized view generation to complete, select both ‘Generate Data’ and ‘Run in Background’. This automatically creates a scheduled task to generate the materialized view. You can check the status of the task under the **Schedule** tab.



If you do not select the 'Generate Data' option, materialized views will be created immediately, but will not be populated with data until the assigned cycle next executes.

11. Press 'Create/Update' to proceed with the options as specified above. If you selected 'Generate Data' only, this will immediately start the generation process. You will be notified upon completion.



If you selected 'Run in Background', the materialized view will be scheduled to begin generation. You can check the status of the task under the **Schedule** tab.

To manage existing materialized views, see *Managing Materialized Views*.

See Also

Distributing Materialized Views (Data Grid), to improve materialization performance.

Managing Materialized Views, to manage existing materialized views. see the

Advanced Dashboard Design, for information about creating Viewsheets.

4.9.2 Incrementally Updating a Materialized View

You can schedule a materialized view to be regenerated on a predefined schedule by assigning a "cycle" to the view. (See *Creating a Materialized View* for more information.) This allows the materialized view to be updated with new data from the database. However, for a large data set, repeatedly regenerating the materialized view in its entirety may be inefficient.

To improve performance, you can elect to update the materialized view *incrementally* based on a set of specified conditions. For example, you can

selectively update the materialized view with data which has posted subsequent to the previous materialization. Likewise, you can delete selected records from the materialized view if they meet a specified condition, such as an expiration date.

You can specify an incremental update condition both for mergeable queries (e.g., JDBC queries) and for non-mergeable queries (e.g., non-JDBC queries, un-parsable queries, etc.), as described below.

Incrementally Updating a Mergeable Query

To specify the criteria that govern the incremental materialized view update for a mergeable query (i.e., standard, parsable, JDBC query), add the desired conditions to the **MV Update** tab of the Data Block in the Data Worksheet. The Data Block's **MV Update** condition will be tested each time an existing materialized view is regenerated (based on the cycle specified for the materialized view), and the materialized view regeneration will be limited to the additions and deletions specified by the condition. (See *Defining a Materialized View Update Condition* in *Advanced Data Mashup* for more information on how to define the update conditions in the **MV Update** tab.)

Incrementally Updating a Non-Mergeable Query

To specify the criteria that govern the incremental materialized view update for a non-mergeable query (e.g., non-JDBC query or un-parsable query), follow the steps below:

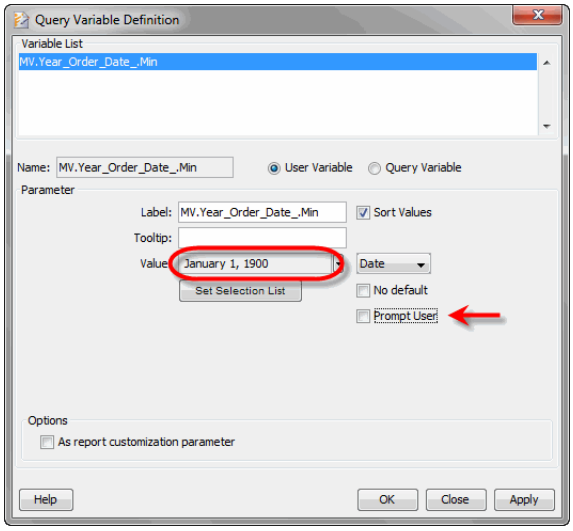
1. Using the Query Wizard in Style Studio, add the desired update condition to the query definition itself. For example:

   ```
   SELECT ... FROM T1 WHERE (...) AND DATE_COL =
   $(MV.LastUpdateTime)
   ```

   The update condition can use the special variables `MV.LastUpdateTime`, `MV.{Column Name}.Min`, and `MV.{Column Name}.Max`, where "`{Column Name}`" is the materialized view column name with non-alphanumeric characters replaced by underscores. For example, a materialized view column named 'Year(Order Date)' should use parameter names `MV.Year_Order_Date_.Min` and `MV.Year_Order_Date_.Max`. (See *Special Parameters for Materialized View Update Conditions* in *Advanced Data Mashup* for more information about these parameters.)

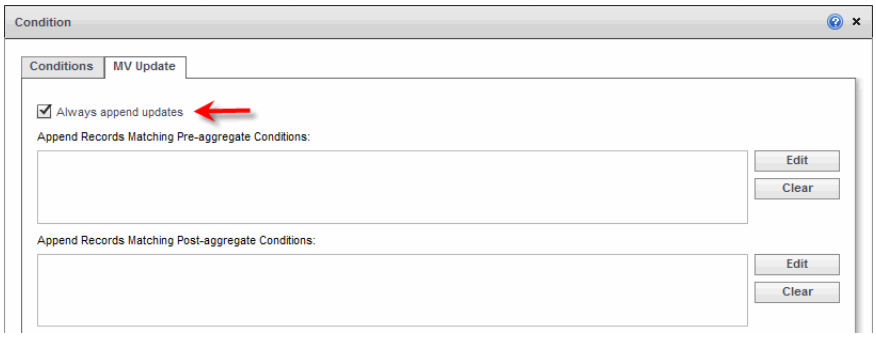
2. Press the 'Parameter' button on the Style Studio toolbar to open the 'Query Variable Definition' dialog box for the query. Disable

parameter prompting and assign default values to the update condition parameters so that the result set is not restricted.



This prevents the user from being prompted for these variables, and ensures that the data is not restricted during the initial materialized view generation. (See *Query Parameters* in *Advanced Data Modeling* for more information about the 'Query Variable Definition' dialog box.)

3. In the **MV Update** tab of the Worksheet Data Block, enable the 'Always append updates' option. (See *Defining a Materialized View Update Condition* in *Advanced Data Mashup* for more information about the **MV Update** tab.)



This ensures that the results of the non-mergeable query will be *appended* to the existing materialized view.

4.9.3 Data Security with Materialized Views

If you are using a VPM to filter data based on particular users, you should generally enable the 'Create Materialized View for all users in group' option. This allows custom materialized views to be generated for each

user (based on their individual VPM permissions). If this option is disabled, only the group name will be available to the VPM, and the filter will not be correctly applied for users.

Implementing Security at the Data Block Level

In some cases, the VPM-based approach described above may generate too many distinct materialized views when security is applied at the user-level. In this case, you can substitute Data Block filtering in place of VPM-based filtering. To implement Data Block filtering, follow the steps below:

1. Place a filtering condition directly on the Worksheet Data Block that supplies data to the Viewsheet. (See *Using a Parameter in a Condition* in *Advanced Data Mashup* for directions.) The condition should include a parameter that filters the Data Block appropriately for a particular user. For example, the condition

   ```
   [First Name][is][equal to][$(_USER_)]
   ```

   filters the Data Block based on the user name by using the predefined variable `_USER_` (which supplies the name of the currently logged-in user). To filter on a property such as the user's corporate customers (i.e., companies), simply define your own variable to store the information:

   ```
   [Company][is][one of][$(comp)]
   ```

   The value of the variable (**`comp`**) will be provided by the SRPrincipal object as explained in the next step.

2. (Optional) Use the *setParameter()* method of the SRPrincipal object to assign a value to the Data Block condition variable. You can do this from within an external module (e.g., an SSO request filter or Login Listener) or from within an internal Viewsheet script.

   a. From within an external module, assign the desired user data to an SRPrincipal parameter of the same name as the target variable. For example, the following Login Listener assigns a set of companies to the '**`comp`**' parameter:

   See *Accessing the User Session with SRPrincipal* in *Advanced Server Integration* for alternate SRPrincipal access.

   ```
   import inetsoft.sree.security.*;

   public class MyLoginListener implements LoginListener {
     public void userLogin(LoginEvent event) {
       SRPrincipal prin = event.getPrincipal();
       prin.setParameter("comp", new String[] {"Ubermeyer",
   "Rutgers Bank", "Bigmart Foodstores"});
     }
   }
   ```

b. From within a Viewsheet script, assign the desired user data to an SRPrincipal parameter of the same name as the target variable. For example, the following script assigns a set of companies to the '**comp**' parameter:

```
var p = parameter['__principal__'];
p.setParameter("comp", new String[] {"Ubermeyer",
"Rutgers Bank", "Bigmart Foodstores"});
```

These procedures pass the desired user data (e.g., companies) directly into the Data Block condition. If a materialized view has been generated for the data set, the condition is applied to the *post-materialized* data set, extracting the appropriate data for the logged-in user. There is therefore no need to generate a distinct materialized view for each user.

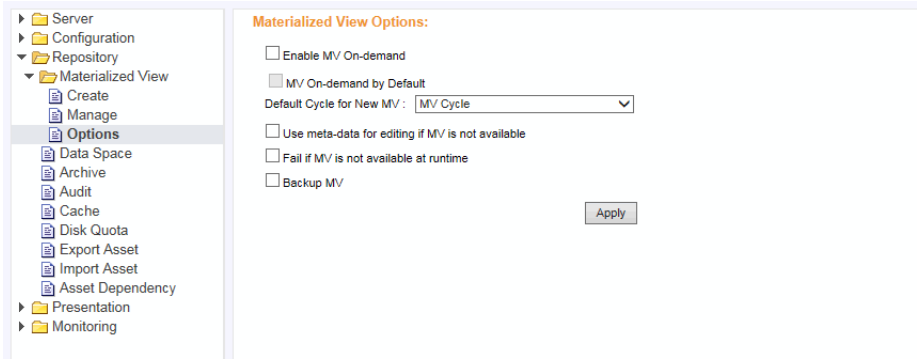
Multi-tenant Considerations

For multi-tenant implementations utilizing the multi-connection feature, materialized views are created based on user/group connection assignments. For this reason, if you set a Viewsheet's permissions by *role* (on the Viewsheet's **Security** tab), you should also add individual permissions for users or groups that have particular connections. This will enable the materialized views to be generated from the appropriate connections.

See Also
Configuring Viewsheets, for information about setting Viewsheet security.
Adding Additional Connections, in *Advanced Data Modeling*, for information on connections.

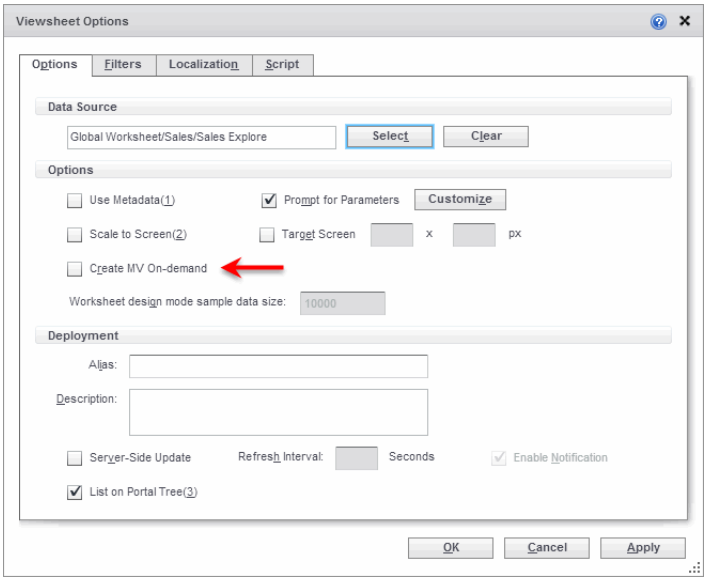
4.9.4 Materialized View Options

To configure options for materialized views, select the 'Repository' > 'Materialized View' > 'Options' page under the **Server** tab.



This page contains the following global options that affect all materialized views.

- **Enable MV On-demand**: Specifies that the 'Viewsheet Options' dialog box in Visual Composer should offer the 'Create MV On-demand' option for individual Viewsheets. When 'Create MV On-demand' is enabled for a Viewsheet, a new materialized view is automatically created on-the-fly if the Viewsheet does not have an existing materialized view (either because the materialized view was never populated with data or because the Viewsheet underwent modifications that rendered an existing materialized view unusable). The materialized view will supply data to the Viewsheet both at design time and at runtime. See *Setting Viewsheet Options* in *Advanced Dashboard Design* for information on setting this option.



- **MV On-demand by Default**: Specifies that the 'Viewsheet Options' dialog box in Visual Composer should have the 'Create MV On-demand' option enabled by default for all new Viewsheets. This relieves the administrator from creating a materialized view for each Viewsheet individually.
- **Default Cycle for New MV:** Specifies the default cycle to be used for on-demand materialized view generation. See *Scheduler Cycle* for information on how to create a cycle.
- **Use meta-data for editing if MV is not available**: Specifies that when a materialized view is not available to supply data for a Viewsheet (for any reason), the design mode of Visual Composer should display meta-data rather than querying the database for data. This will improve design-time performance. See *Suppressing Query Execution in Design View* in *Advanced Dashboard Design* for more information about meta-data mode.
- **Fail if MV is not available at runtime**: When a user loads a Viewsheet in the User Portal, this option generates an error message if the

requested Viewsheet does not possess a corresponding materialized view or the materialized view fails to generate (due to user cancellation or for any other reason). If you prefer the Viewsheet to query the database for data rather than generate an error in this case, disable 'Fail if MV is not available at runtime'.

- **Backup MV**: Specifies that a back-up should be automatically created for any materialized view whenever the materialized view is regenerated, i.e., whenever the associated cycle is executed. In the event that a materialized view fails to generate, this allows you to restore a prior functional materialized view from back-up rather than regenerating the entire materialized view. To create a backup on-demand or restore a materialized view from back-up, see *Managing Materialized Views*.

4.9.5 Materialized View Performance Considerations

When you design Data Worksheets and Viewsheets (dashboards) to be used with materialized views, keep the following points in mind:

- In the Data Worksheet, try to keep the design as simple as possible.
- In the Data Worksheet, if you are joining or concatenating tables, try to place any parameterized filtering conditions (i.e., conditions that change in response to user input or the runtime environment) on the *resultant* Data Block rather than on the base tables. Conditions that are placed on base tables may necessitate the creation of a "sub-materialized view" corresponding to each base table and require additional post-processing of these sub-materialized views.
- In the Viewsheet, try to place selection (filtering) components such as Selection List and Range Slider on the *resultant* Data Block of the Data Worksheet (i.e., the final Data Block resulting from join or concatenation operations) rather than on the base tables.
- In the Viewsheet, try to avoid creating tables that display large numbers of records (i.e., long, scrolling tables).
- If you wish to add a fixed filter condition (i.e., a condition that does *not* change in response to user input or the runtime environment), implement the fixed condition in the Data Worksheet rather than in the Viewsheet.

See *Transformations and Faults* for an approach to addressing other common impediments to effective materialization.

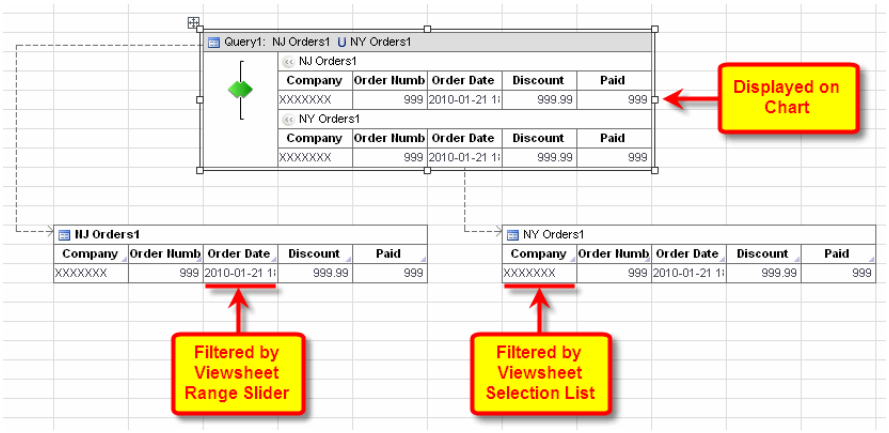
See Also

Transformations and Faults, for more information about designing Worksheets to facilitate materialization.

4.9.6 Transformations and Faults

When a particular data view component (Chart, Table, etc.) is materialized, the materialization process *caches* the data in the underlying Worksheet Data Block to which the component is bound. This allows the component to rapidly query the cached dataset rather than the database.

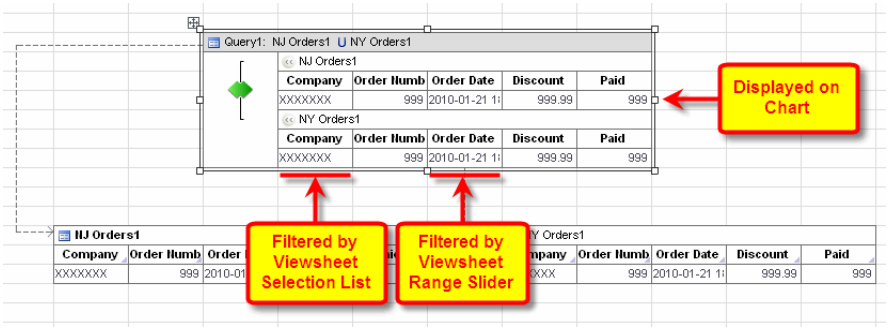
In order for the underlying Data Block to be materialized, any Viewsheet selections (e.g., Selection Lists) that filter the Data Block must be applied directly to the Data Block itself, rather than to any precursor data blocks. For example, consider the following Worksheet:



This Worksheet is used by a Viewsheet that has the following component bindings:

- Chart (data view component) bound to the resultant ‘Query1’ Data Block
- Range Slider bound to the ‘NJ Orders1’ Data Block
- Selection List bound to the ‘NY Orders1’ Data Block

To materialize the ‘Query1’ Data Block for use by the Chart, the materialization process must first *transform* the Viewsheet by “moving up” the selection components to operate on the resultant Data Block, as shown below:



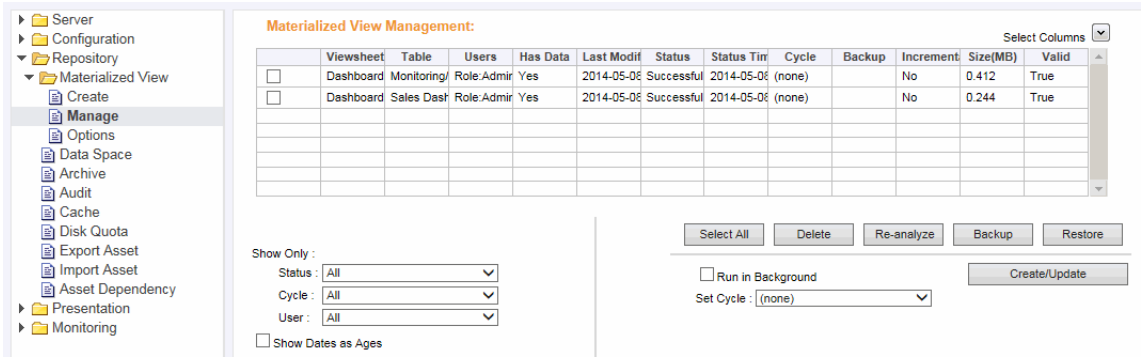
The automatic transformation is internal, and does not alter the actual Viewsheet.

This transformation process is done internally and automatically during the materialization process. The result of the transformation is a new representation that preserves the same data as the original Data Block but is more appropriate for querying. However, when the transformation cannot be accomplished in a way that preserves the data intact, Enterprise Manager displays a “fault” in the ‘Optimize Plan’ dialog box. The fault means that the Data Block in question cannot be materialized.

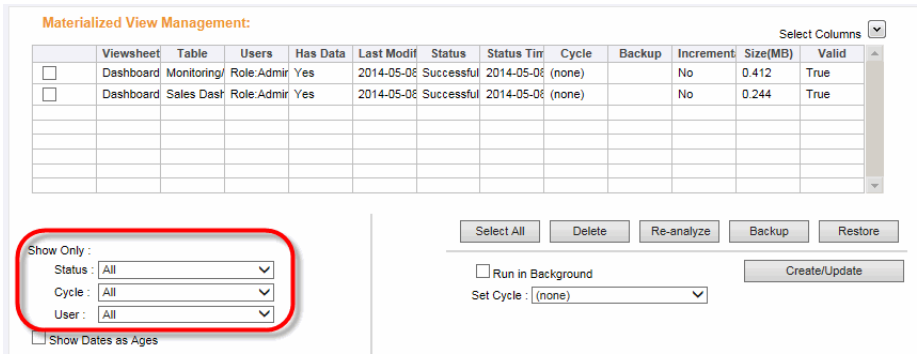
In many of these instances, you may be able to facilitate materialization by “moving the selection up.” To do this, evaluate the placement of your selection components, and try to relocate them (i.e., move them up) to filter the *resultant* Data Block rather than the precursor Data Blocks. By doing this you may enable Enterprise Manager to materialize the problem Data Block.

4.9.7 Managing Materialized Views

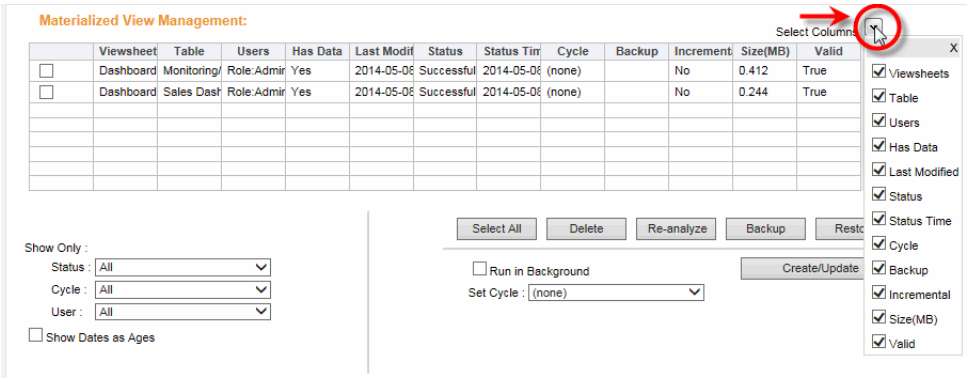
Manage your existing materialized views from the ‘Repository’ > ‘Materialized View’ > ‘Manage’ page under the **Server** tab.



The table lists all existing materialized views. To restrict the items shown in the table by status, cycle, or user, make the desired selections from the left-side menus:



To restrict the columns shown in the table, expand the ‘Select Columns’ menu on the right side. Choose the columns you want to display, and press ‘X’ to close the menu. For information on the available columns, see *Materialized View Table Column Description*.



To select all materialized views in the table, press ‘Select All’.

To manually back-up a materialized view, select the desired materialized view in the table and press ‘Backup’.

To revert a materialized view to an existing back-up version, select the desired materialized view in the table and press ‘Restore’.

To delete a materialized view (for example, if changes have been made to the underlying database schemas), select the materialized view in the table and press ‘Delete’.

See Also

Creating a Materialized View, for information on materialized view creation.

Advanced Dashboard Design, for more information about creating and editing Viewsheets.

Materialized View Table Column Description

The following table columns are available on the ‘Materialized View Management’ page.:

COLUMN	DESCRIPTION
Viewsheets	Viewsheets using the materialized view.
Table	Data block in the Data Worksheet that is materialized.
Users	Users of the materialized view.
Has Data	Indicates that the materialized view has been populated with data.
Last Modified	Time at which the materialized view was most recently updated. To display modification dates as ages (e.g., “20 minutes”), enable ‘Show Dates as Ages’.

COLUMN	DESCRIPTION
Status	Current state of the materialized view: ‘Successful’, ‘Failed’, ‘Pending’, or ‘Generating’. ‘Pending’ indicates that the materialized is scheduled to generate automatically on a cycle.
Status Time	Timestamp of the last status update. To update the status, reload the page.
Cycle	Cycle on which the materialized view is scheduled to generate. Use the ‘Set Cycle’ menu to change the cycle. See *Scheduler Cycle* for information about how to create a cycle.
Backup	Timestamp on the materialized view back-up. Use the ‘Restore’ button to restore from back-up.
Incremental	Indicates that the materialized view has an incremental update condition. See *Incrementally Updating a Materialized View* for information on how to set update conditions.
Size (MB)	The combined size of all of the materialized view data files, no including the back-up.
Valid	Indicates that the current materialized view is compatible with the Viewsheet. If the materialized view is no longer valid (due to subsequent Viewsheet modifications), select the invalid materialized view in the table. Press ‘Re-analyze’ and then press ‘Create/Update’. See *Creating a Materialized View* for more information about analyzing a Viewsheet.

4.9.8 Distributing Materialized Views (Data Grid)

To improve Viewsheet performance, you can distribute a materialized view across multiple machines. This distributed configuration is called a *data grid*. In a data grid, the materialized dataset is saved on separate physical machines (nodes) in the grid. This allows the Viewsheet server to pull small blocks of data from each of the nodes, thereby distributing the processing load and minimizing hard disk access for any particular machine. This increases total throughput and improves Viewsheet responsiveness.

A data grid is completely transparent to the end-user. Users can access their Viewsheets in the same manner regardless of whether a data grid has been configured or not. However, Viewsheet performance will in general be better with a data grid in place.

A data grid configuration consists of the following components:

- **Report Server**: The Report Server is a J2EE web application that handles all user requests for reports and Viewsheets in the repository. The Report Server also provides access to the Enterprise Manager for managing the repository and the Scheduler application. For Viewsheets that utilize materialized views, the Report Server delegates data retrieval to the *Data Server.*
- **Data Server**: The Data Server is a J2EE web application that handles materialized view requests from the Report Server and dispatches tasks

to the *Data Nodes*. The Data Server also provides access to the Enterprise Manager for monitoring data grid status and managing materialized views. The Data Server requires an ‘H’ or ‘G’ key.

```
Note: Each Data Node must be a distinct machine.
```

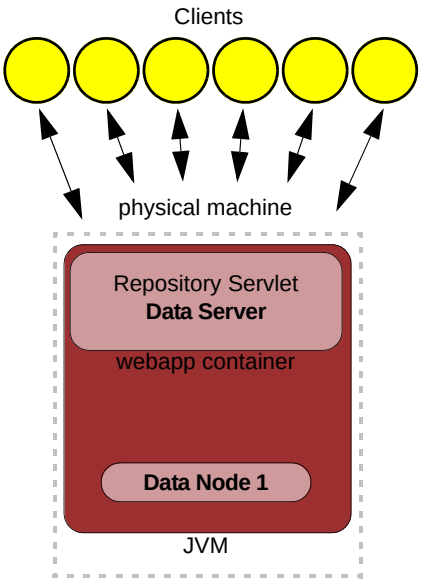
- **Data Node**: A Data Node is a machine running the Data Node service, a stand-alone Java process that has no user interface and does not require a web application container. The Data Nodes store data from the materialized view and respond to requests for data from the Data Server, (via socket connections). A Data Node requires an ‘H’ or ‘G’ key.

You can configure the Report Server, Data Server, and Data Node in various ways, as described in the following sections. Note that the Data Server and a Data Node *cannot* reside on the same machine, except in the default installation (*Report Server, Data Server, and Data Node on Same Machine (Default)*).

Report Server, Data Server, and Data Node on Same Machine (Default)

The default Style Intelligence installation procedure installs a Report Server, which includes a Data Server and single Data Node. If you do not explicitly configure a multiple-machine data grid (as described in later sections), the Report Server automatically starts the Data Server and Data Node on the same machine.

Figure 24. Report Server, Data Server, and Data Node on one machine



Viewsheet materialization with this default configuration generally delivers an improvement in performance over non-materialized Viewsheets. If performance is satisfactory, you do not need to further modify any settings.

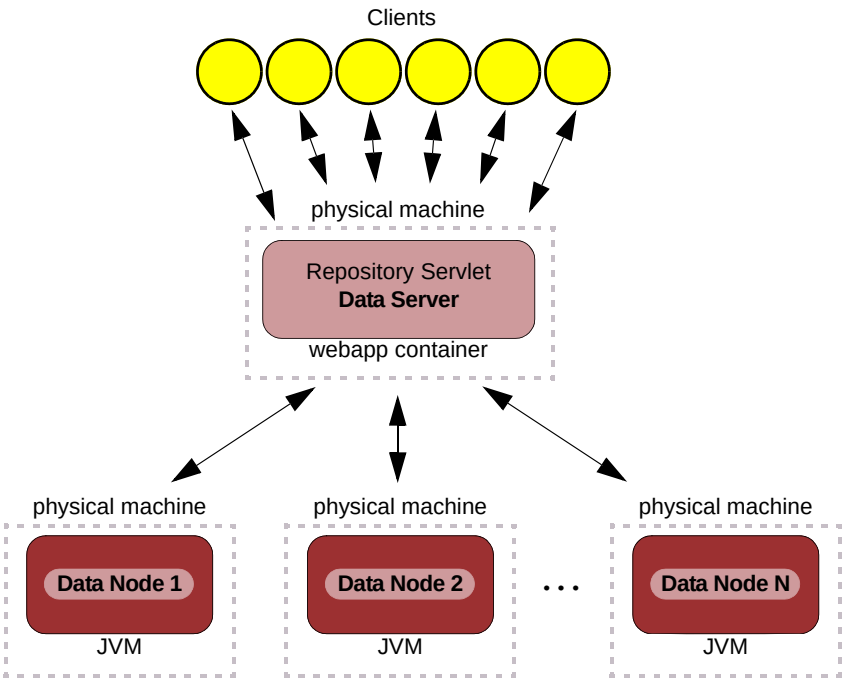
However, you can generally obtain better performance for materialized Viewsheets by distributing the materialized view across multiple machines, as described in the following sections.

To create a materialized view using Enterprise Manager, see *Creating a Materialized View*.

Report and Data Server on Machine 1, Data Nodes on Machines 2-N

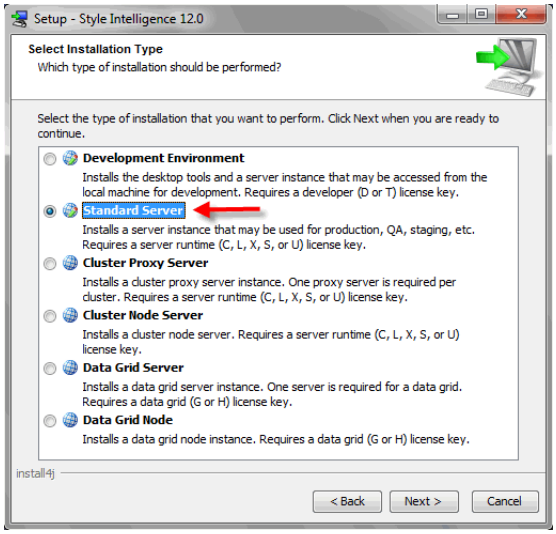
You can install the Report Server and Data Server on a single machine, and distribute the Data Nodes across a different set of machines.

Figure 25. Report and Data Server on Machine 1, Data Nodes on Machines 2-N



To configure the data grid this way, follow the steps below:

1. Install the Report Server on the desired machine by using the ‘Standard Server’ option of the Style Intelligence installer.



Note: You must access Enterprise Manager in order for the Data Nodes to connect successfully.

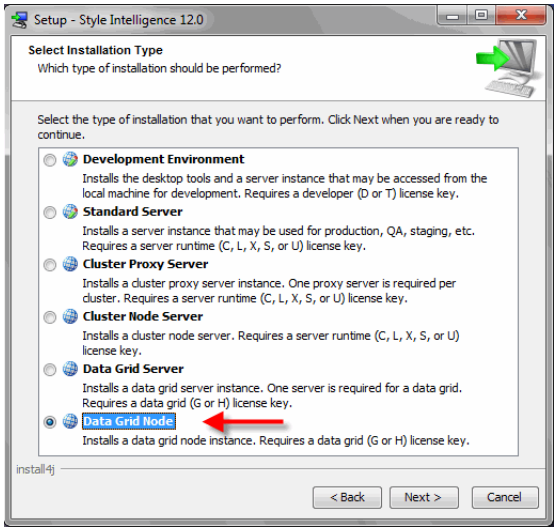
2. Start the Style Intelligence server. In a browser, log into the Enterprise Manager.

3. For each Data Node machine, configure the node by following the steps below:

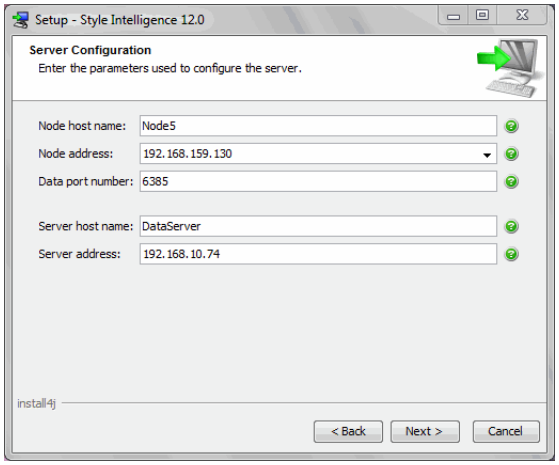
    a. Run the Style Intelligence installer on the machine that will host the Data Node.

    b. On the ‘License Key’ page, enter the appropriate “G” key or “H” key.

    c. On the ‘Installation Type’ page, select the ‘Data Grid Node’ option.



d. On the ‘Server Configuration’ page, set the ‘Node host name’ and ‘Node address’ as appropriate for the machine onto which you are installing the Data Node. Set the ‘Server host name’ and ‘Server address’ to the name and address of the machine hosting the Data Server.



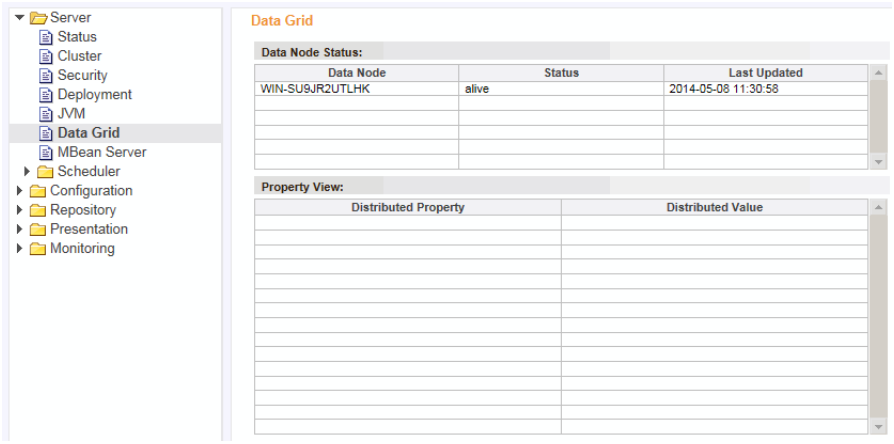
e. Follow the rest of the on-screen installation instructions to complete the installation of the Data Node.

f. Launch the Data Node from the Windows *Start* menu. (Alternatively, execute `bin/DataGrid.exe`.)

This will open a command window. After a few moments, the command window should indicate a successful connection to the Report/Data Server machine.

4. On the Report Server machine, return to Enterprise Manager. Open the ‘Server’ > ‘Data Grid’ page under the **Server** tab. Verify that

the Data Node machine you have started is shown with “alive” status.

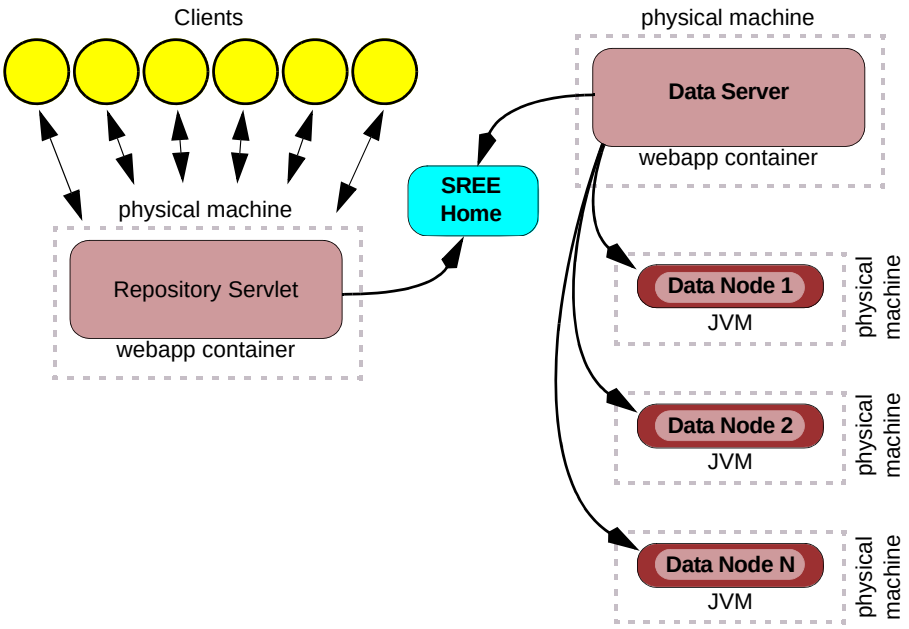


To create a materialized view in this data grid environment, use Enterprise Manager on the Report/Data Server machine. See *Creating a Materialized View*.

Report Server on Machine 1, Data Server on Machine 2, Data Nodes on Machines 3-N

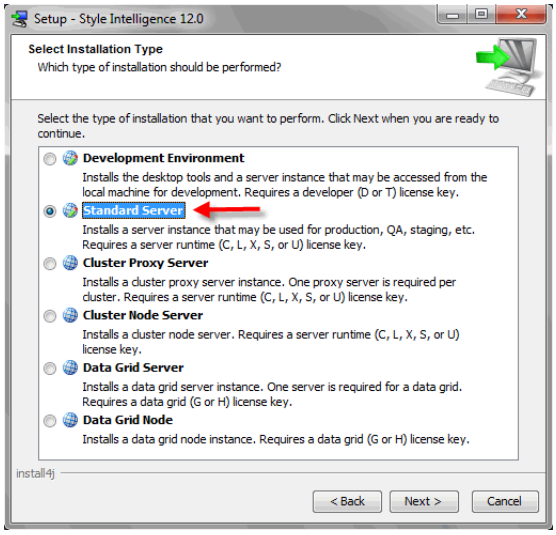
You can isolate each component of the data grid so that the Report Server, Data Server, and each Data Node reside on an independent machine. Note that the Report Server and Data Server must still access a shared SREE Home repository location.

Figure 26. Report Server on Machine 1, Data Server on Machine 2, Data Nodes on Machines 3-N



To configure the data grid this way, follow the steps below:

1. Install the Report Server on the desired machine by using the 'Standard Server' option of the Style Intelligence installer.



`The Report Server SREE Home directory must be accessible to the Data Server.`

2. Set *sharing* on the SREE Home directory (default: `sree/WEB-INF/classes`) so that it can be accessed from another machine.

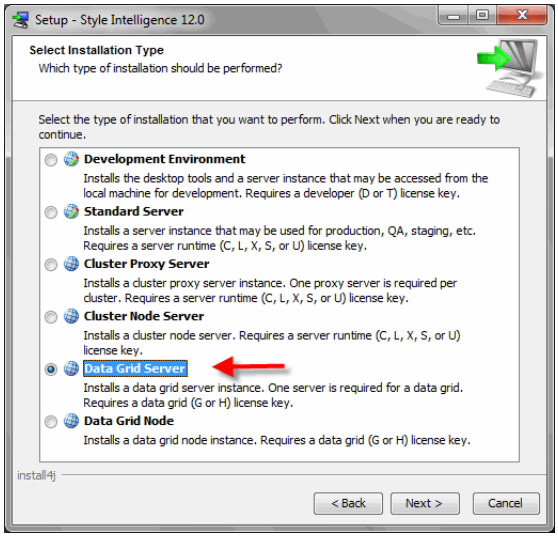
3. If the Report Server is running, stop it now.

4. Install the Data Server on the desired machine. Follow the steps below:

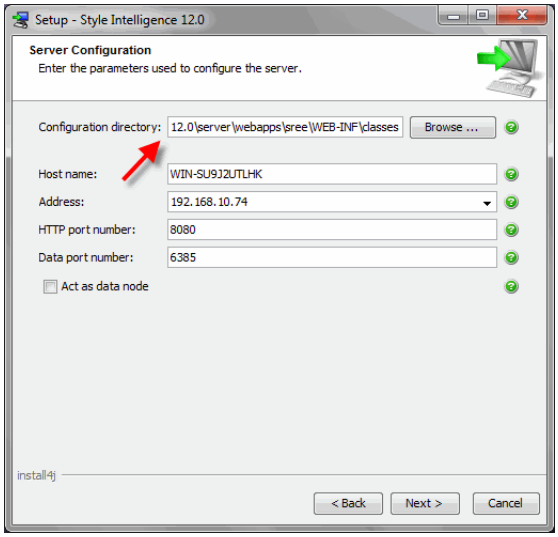
   a. Run the Style Intelligence installer on the machine that will host the Data Server.

   b. On the ‘License Key’ page, enter the appropriate “G” key or “H” key.

   c. On the ‘Installation Type’ page, select the ‘Data Grid Server’ option.



   d. On the ‘Server Configuration’ page, set the ‘Registry directory’ to the SREE Home directory that you shared in a previous step. Set the ‘Host name’ and ‘Address’ as appropriate for the machine onto which you are installing the Data Server.



e. Follow the rest of the on-screen installation instructions to complete the installation.

   In addition to installing the Data Server onto the target machine, the installer also adds a number of properties to the `sree.properties` file in the shared SREE Home directory. Among the properties that are automatically added are the following: `comm.port`, `comm.this.host`, `comm.{machineName}.ips`, `fs.server`.

5. Make certain that any required database drivers are present in the `sree/WEB-INF/lib` directory of the Data Server.

6. Launch the Data Server from the Windows *Start* menu. (Alternatively, execute `bin/InetsoftServer.exe`.) There may be a slight delay before the Data Server starts.

7. On the Report Server machine, launch the Style Intelligence Server from the Windows *Start* menu.

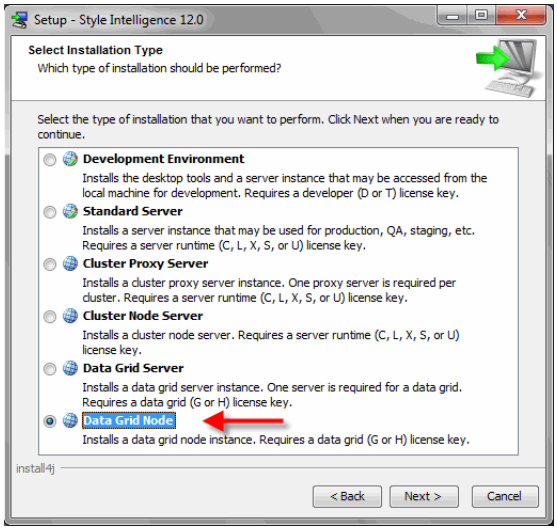
8. Log into Enterprise Manager on the Report Server machine.

9. For each Data Node machine, configure the node by following the steps below:

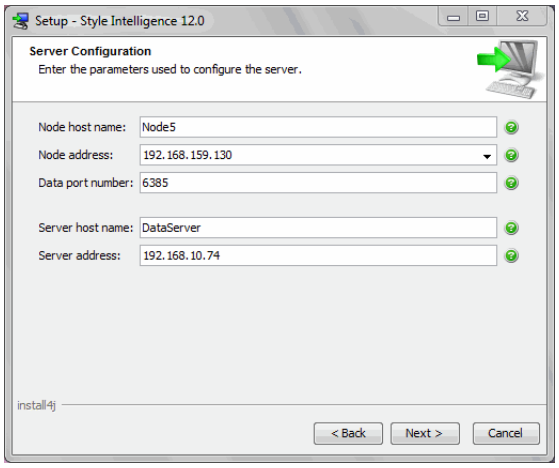
   a. Run the Style Intelligence installer on the machine that will host the Data Node.

   b. On the 'License Key' page, enter the appropriate "G" key or "H" key.

c. On the ‘Installation Type’ page, select the ‘Data Grid Node’ option.



d. On the ‘Server Configuration’ page, set the ‘Node host name’ and ‘Node address’ as appropriate for the machine onto which you are installing the Data Node. Set the ‘Server host name’ and ‘Server address’ to the name and address of the machine hosting the Data Server (*not* the Report Server machine).



e. Follow the rest of the on-screen installation instructions to complete the installation of the Data Node.

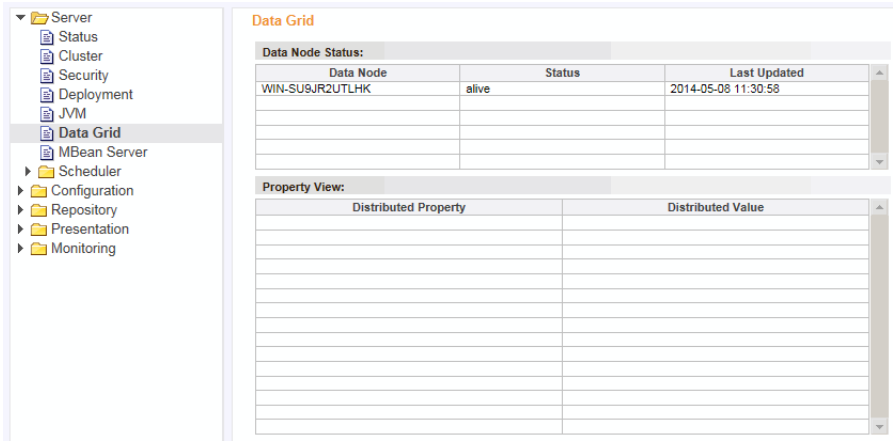
f. Launch the Data Node from the Windows *Start* menu. (Alternatively, execute `bin/DataGrid.exe`.)

This will open a command window. After a few moments, the command window should indicate a successful connection to the Data Server machine.

Enterprise Manager on the Report Server machine cannot display node status.

10. Open Enterprise Manager on the Data Server machine (*not* the Report Server machine). Open the 'Server' > 'Data Grid' page under the **Server** tab. Verify that all Data Nodes you have previously started are shown with "alive" status.

Figure 27. The Data Grid page.



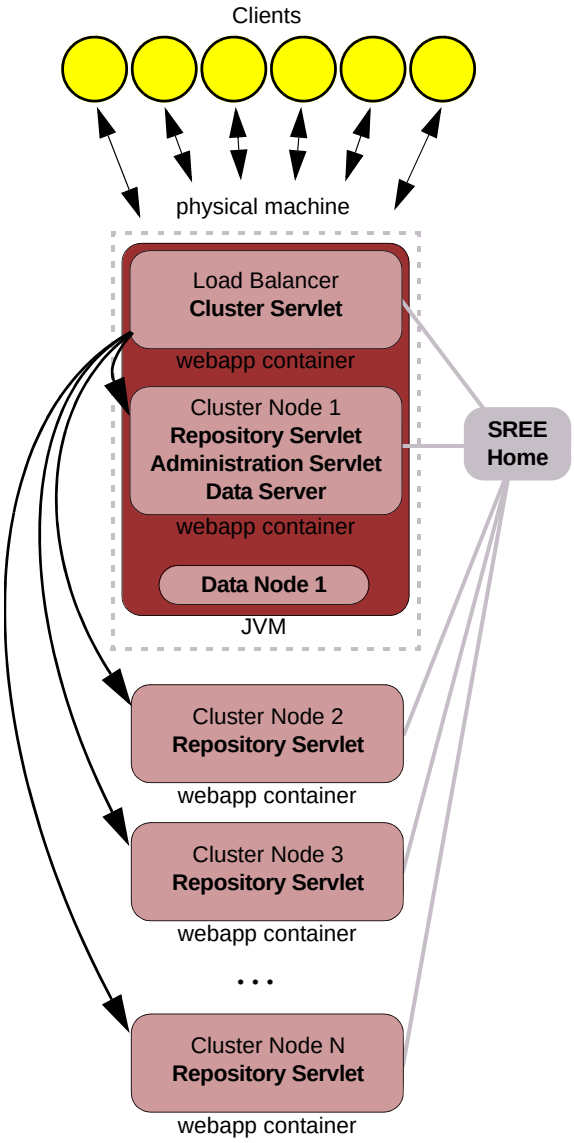
To create a materialized view in this data grid environment, you must log into Enterprise Manager on the Data Server machine. (You cannot create a materialized view by using Enterprise Manager on the Report Server machine.) See *Creating a Materialized View* for further information.

Default Materialization in a Clustered Environment

The default installation for a clustered reporting environment is the same as that for a non-clustered environment, described in *Report Server, Data Server, and Data Node on Same Machine (Default)*. See *Configuring Server Clustering* for information about setting up a reporting cluster.

In this default configuration, the Report Server, Data Server, and Data Node are installed on a single machine. If you do not explicitly configure a multiple-machine data grid (as described in *Distributed Materialization in a Clustered Environment*), the Report Server starts the Data Server and Data Node on the same machine.

Figure 28. Report Server, Data Server, and Data Node on one machine, in clustered environment

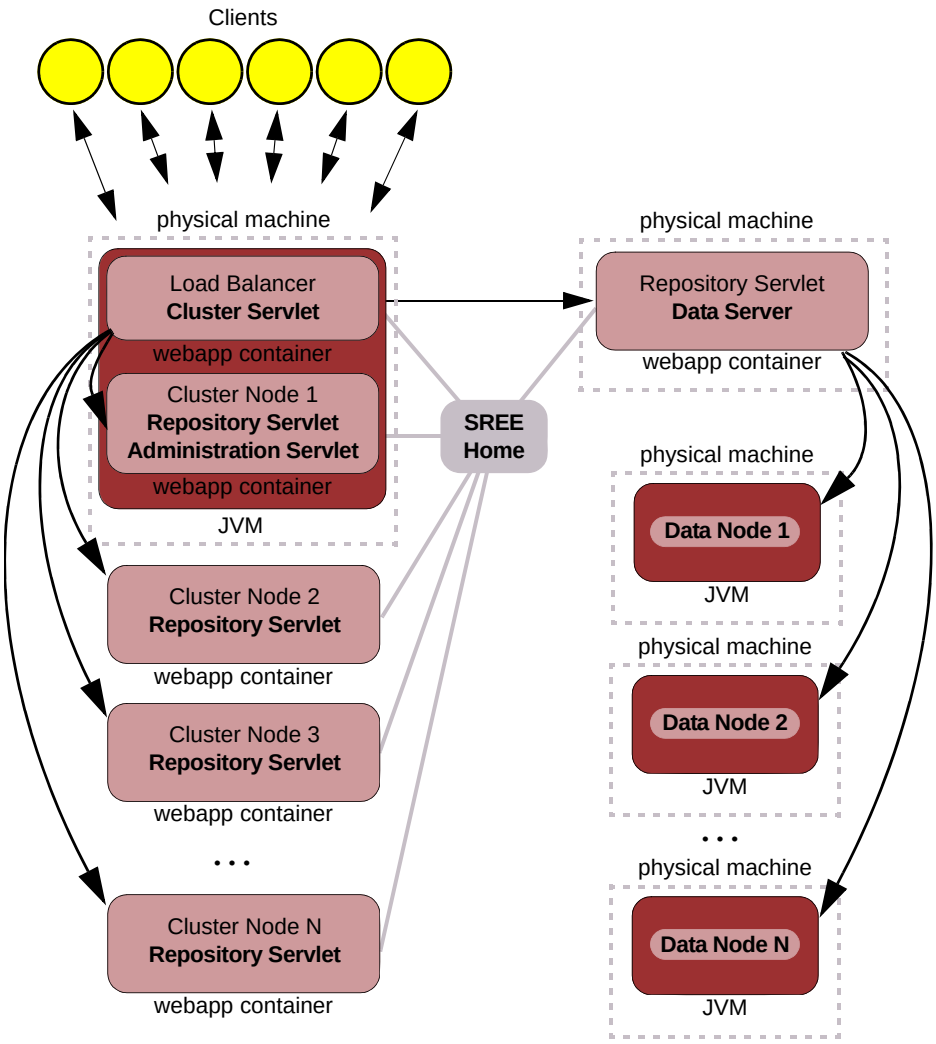


To obtain better performance, distribute the materialized view across multiple machines as described in *Distributed Materialization in a Clustered Environment*. To create a materialized view using Enterprise Manager, see *Creating a Materialized View*.

Distributed Materialization in a Clustered Environment

If you are using a clustered environment for report generation (as explained in *Configuring Server Clustering*), and you want to distribute your materialized views across multiple data nodes, use independent machines for each cluster node and each data node, as shown below.

Figure 29. Data Grid in a Clustered Environment



To configure a data grid in a clustered environment, follow the steps below:

1. Follow the instructions in *Configuring Server Clustering* to created a clustered reporting environment.

2. Follow the instructions in *Report Server on Machine 1, Data Server on Machine 2, Data Nodes on Machines 3-N* to create a data grid. However, note the following:

   Do not install the Report Server as described in step 1. The machine which you have already configured in the reporting cluster as “Cluster Node 1” (containing the Administration Servlet) will act as the Report Server for the data grid.

   Do not install the Data Server or Data Nodes on any machine which is already being used as a node in the reporting cluster.

3. Verify that the cluster proxy, cluster nodes, and Data Server are all utilizing the same SREE Home directory. This allows the cluster nodes and the Data Server to share the same materialized view and security settings, so that a materialized query created by a cluster node can be executed on the Data Server.

To create a materialized view in this data grid environment, you must log into Enterprise Manager on the Data Server machine. (You cannot create a materialized view by using Enterprise Manager on any other machine.) See *Creating a Materialized View* for further information.

Troubleshooting

If one or more Data Nodes are not accessible to the Data Server (i.e., the Data Nodes are not listed on the 'Data Grid' page in Enterprise Manager on the Data Server machine), check the following settings.

- The Report Server and Data Server must share the same SREE Home directory. (See *Specifying the Repository Directory* to change the SREE Home location used by a Style Intelligence server.) Make sure that "sharing" is set appropriately so that this folder is accessible to both servers.
- In the *sree.properties* file within the shared SREE Home directory, the `fs.local` property should be set to `false` if you are using a distributed materialized view. The `fs.local` property should be set to `true` only in the following cases:
  - *Report Server, Data Server, and Data Node on Same Machine (Default)*
  - *Default Materialization in a Clustered Environment*
- In the *sree.properties* file within the shared SREE Home directory, the `fs.server` property should specify the host name of the Data Server.
- In the *sree.properties* file within the shared SREE Home directory, the `fs.nodes` property should specify the host name of all the Data Nodes, separated by spaces.

The settings in the *sree.properties* file are generally configured automatically by the installer. However, if you discover that the `fs.local`, `fs.server`, or `fs.nodes` properties are not assigned the correct values in the *sree.properties* file, enter the correct values by manually editing the *sree.properties* file.

5 Security

Security is an important part of any enterprise application. InetSoft's security model provides security at both a component and functional level. InetSoft's security model also provides security at a very granular object level, e.g., a user may not have access to all the data models or queries in the data source and query registries (*datasource.xml* and *query.xml*). The security model is highly adaptable and configurable, e.g., you can import your existing users/groups/roles from a database or an LDAP server without redefining them in InetSoft's software.

There are two fundamental components to setting up security:

1. Specifying a Security Provider and optionally defining users, groups and roles.

2. Specifying permissions for reports and other components.

5.1 Specifying a Security Provider

A security provider is a module which handles security operations for the report server. You can specify independent security providers to provide the following functions:

You must specify a security provider in order to generate server audit records.

- **Authentication**: Retrieving users, roles, and groups; authenticating logins.
- **Authorization**: Storing and retrieving permissions for different components and functions.

See Also
Selecting a Security Provider, in *Advanced Server Integration*, for information on choosing a security configuration.

5.1.1 The Security Provider Page

To administer security provider options, select the 'Server' > 'Security' option under the **Server** tab.



Select 'Enable Security' to activate security settings.

In the ‘Authentication’ menu, select the desired method for verifying user login credentials. In the ‘Authorization’ menu, select the desired method for assigning asset and activity permissions.

Enterprise Manager provides a default security provider to handle authentication, creation of users/groups/roles, and the storage of component permissions. For information on using the default security provider for authentication and/or authorization, see *Configuring the Default Security Provider*.

If your users/groups/roles have been pre-defined in an LDAP server (e.g., Active Directory), Enterprise Manager can access the existing LDAP data for authentication and/or authorization. For more information, see *Configuring an LDAP Security Provider*.

You can also create a custom security provider as a Java class that implements the ‘Security Provider’ interface, and which can be used when retrieving users/groups/roles from a database. See *Custom Security Integration* for more details.

See Also

Selecting a Security Provider, in *Advanced Server Integration*, for information on choosing a security configuration.

5.1.2 Configuring the Default Security Provider

The default security provider is a built-in implementation of the InetSoft security model, and does not require any special configuration.

To use the default provider for authentication (i.e., verifying user credentials), select the ‘Default’ option from the ‘Authentication’ menu on the ‘Security Provider’ page. (See *Specifying a Security Provider*.) User/group authentication information is accessed from the *password.xml* file.

To use the default provider for authorization (i.e., assigning permissions and access), select the ‘Default’ option from the ‘Authorization’ menu on the ‘Security Provider’ page. (See *Specifying a Security Provider*.) Authentication information is accessed from the *acl.xml* file. Roles are stored in the *role.xml* file.

Note: When using the ‘Default Security Provider’, the default login/password to the Enterprise Manager is admin/admin.

When you select the default security provider for authorization, you can use the **Users** tab of Enterprise Manager to create users/groups/roles and associate these with components and objects managed by the server. See the *Users, Roles, and Groups* and *Setting Permissions* sections for details on how to set up users/groups/roles and assign permissions.

5.1.3 Configuring an LDAP Security Provider

The LDAP security provider is useful when your users, groups, and roles are already defined in an LDAP environment such as Active Directory, Sun ONE (iPlanet), or any generic LDAP implementation. You can use the LDAP server for either authentication or authorization, although LDAP is not generally recommended for authorization.

```
LDAP is recommended for authentication but not for authorization
because authorization requires modification of the LDAP schema.
```

To use the LDAP security provider for *authentication* (i.e., verifying user credentials), select the 'LDAP' option from the 'Authentication' menu on the 'Security Provider' page. (See *Specifying a Security Provider*.) All user, group, and role information is retrieved from the LDAP server and cannot be created or configured in Enterprise Manager. To configure the LDAP connection, see *Defining the LDAP Schema*.

To use the LDAP security provider for *authorization* (i.e., assigning permissions and access), select the 'LDAP' option from the 'Authorization' menu on the 'Security Provider' page. (See *Specifying a Security Provider*.) Authentication information is then stored and accessed from the LDAP server (with modification of the LDAP schema). See *Writing to the Active Directory Schema* for information on making the Active Directory schema writable.

See Also
Selecting a Security Provider, in *Advanced Server Integration*, for information on choosing a security configuration.

Defining the LDAP Schema

The structure of the LDAP schema must be defined in the Enterprise Manager. To add schema information, follow the steps below:

1. Select the **Server** tab, and then select the 'Server' > 'Security' option to open the 'Security Provider' page.

```
LDAP is recommended for authentication but not for authorization
because authorization requires modification of the LDAP schema.
```

2. Select the 'LDAP Security Provider' option for 'Authentication' or 'Authorization', as desired.

3. If you use a back-up LDAP server to provide security in the event of a primary server failure, check 'Enable Backup Server'.

4. Select 'Main server' from the 'Enable Backup Server' menu.

5. Choose the appropriate LDAP server implementation from the ‘LDAP Server’ menu. Make sure the main LDAP server is correctly installed and configured, and supply the necessary connection information. See *Sun ONE (iPlanet) Server*, *Active Directory Server*, and *Generic LDAP Server* for more details.

6. Select ‘Backup server’ from the ‘Enable Backup Server’ menu.

7. Choose the appropriate LDAP server implementation from the ‘LDAP Server’ menu. Make sure the backup LDAP server is correctly installed and configured, and supply the necessary connection information. See *Sun ONE (iPlanet) Server*, *Active Directory Server*, and *Generic LDAP Server* for more details.

8. Enable ‘Search User Subtree’ to perform a recursive search on the the specified user base.

9. Press the ‘Test Connection’ button to test the connection to the LDAP database.

10. In the ‘EM Administrator Role’ field, enter the role that a user must possess in order to log into Enterprise Manager.

11. Press ‘Test EM Login’ to verify credentials for logging into Enterprise Manager.

12. Press ‘Apply’ to save the settings.

Once the directory server environment is configured, you can assign permissions to users/groups/roles for different components and functions. (See *Setting Permissions* for more information.)

Notes on LDAP Security Providers

- Since directory servers do not currently support roles fully, groups in the LDAP schema are mapped to roles in Style Intelligence.
- If the administrator password on the LDAP server is changed, you will not be able to log into Enterprise Manager in the usual way. Instead, log into Enterprise Manager by using the administrator’s Distinguished Name (DN) together with the new LDAP administrator password. This allows Enterprise Manager to connect to the LDAP server and update the security configuration. You can then log into Enterprise Manager using the usual administrator credentials.
- Enterprise Manager authentication is integrated with Report Server authentication. Therefore, when you enable an LDAP security provider authentication you can no longer log into the EM using the default ‘admin/admin’ credentials. Instead, you must log in as a user with an

'Administrator' role (group). If there is no role (group) named 'Administrator' in your system, you can set the role by adding the '`role.administrator`' property in the *sree.properties* file, e.g., `role.administrator=SRAdmin`.

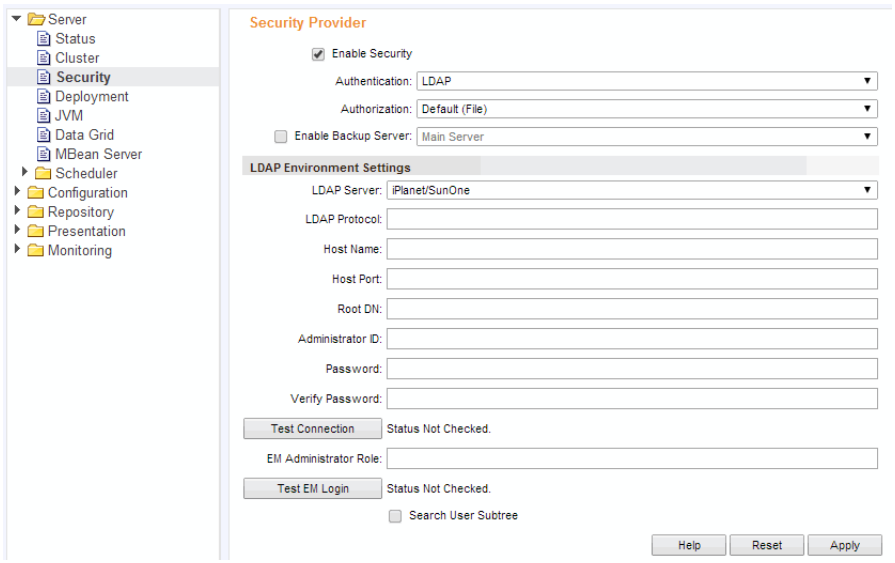
- The performance of any *AbstractSecurityProvider* security implementation (including LDAP security) can be enhanced by setting '`security.cache=true`' in the *sree.properties* file.
- Security data is cached. To refresh the cache when security data changes, press the 'Clear Security Cache' button on the 'Performance Options' page. See *Tailoring Server Performance* for more information.

The following sections provide a detailed description of how to set up the different types of LDAP implementation supported by InetSoft products.

Sun ONE (iPlanet) Server

To use the Sun ONE Directory Server, select 'iPlanet/SunOne' from the 'LDAP Server' menu under the 'Security Provider' page. (See *Specifying a Security Provider* information about the 'Security Provider' page.)

Figure 30. LDAP Environment Setup for Sun ONE (iPlanet)



When using the Sun ONE Directory Server, you must set up a base DN (Distinguished Name, e.g., dc=inetsoft, dc=com) for the company. In addition, you should configure the server by setting up access permissions for this DN. Finally a schema must be created to enable access to deployed reports via Sun ONE.

When the Sun ONE (iPlanet) Directory Server is correctly installed and configured, you must next create a report schema. To create the schema, follow these steps:

1. Start the Sun ONE server console, expand the node under root that corresponds to the DN, e.g., ‘inetsoft’ (in the left pane), then expand the node ‘Server Group’.

2. Click on ‘Directory Server’ in the left pane, then click the ‘Open’ button in the right pane.

3. Open the Directory Server console, click the **Configuration** tab, and highlight the node ‘schema’ under the root of ‘hostname.inetsoft:389’, where ‘hostname’ is your machine name and ‘inetsoft’ is your DN.

4. In the right pane, select the **Attributes** tab.

5. To add a new attribute, click the ‘Create’ button on the bottom of the page.

6. Fill all mandatory fields in the pop up dialog titled ‘Create Attribute’ then click ‘OK’.

7. Repeat Steps 4 and 5 to add all attributes listed in the Table 7 below.

8. To add a new objectclass, select the **Object classes** tab at the top of the screen, then click the ‘Create’ button at the bottom of the screen.

9. In the ‘Name’ field, type “inetsoftreplet”, then select ‘cn’ from the ‘Available’ attributes list and move it to the ‘Required’ attributes list. Continue to move all attributes you created in Step 7 from the ‘Available’ attributes list to the ‘Required’ attributes list, then click ‘OK’

10. Go back to the **Object classes** tab. Object class ‘inetsoftreplet’ should be in the object classes list

Table 7. Example Schema for Sun ONE Directory Server

- objectclass inetsoftreplet
- dn: ‘cn=charts, dc=inetsoft’

- RDN: ‘cn=charts’

ATTRIBUTE	DESCRIPTION	CLASS DEFINED	DATA TYPE
cn	report name	top	Directory String
readusers	Users have ‘read’ permission	inetsoftreplet	Directory String(multi-valued)
writeusers	Users have ‘write’ permission	inetsoftreplet	Directory String(multi-valued)
deleteusers	Users have ‘delete’ permission	inetsoftreplet	Directory String(multi-valued)
readgroups	Groups have ‘read’ permission	inetsoftreplet	Directory String(multi-valued)
writegroups	Groups have ‘write’ permission	inetsoftreplet	Directory String(multi-valued)
deletegroups	Groups have ‘delete’ permission	inetsoftreplet	Directory String(multi-valued)
default	If this report is default no / yes	inetsoftreplet	Directory String(multi-valued)

- objectclass inetsoftreplet
  oid inetsoftreplet-oid
  superior top

- requires
  cn,

- allows
  readusers,
  writeusers,
  deleteusers,
  readgroups,
  writegroups,
  deletegroups,
  default

Active Directory Server

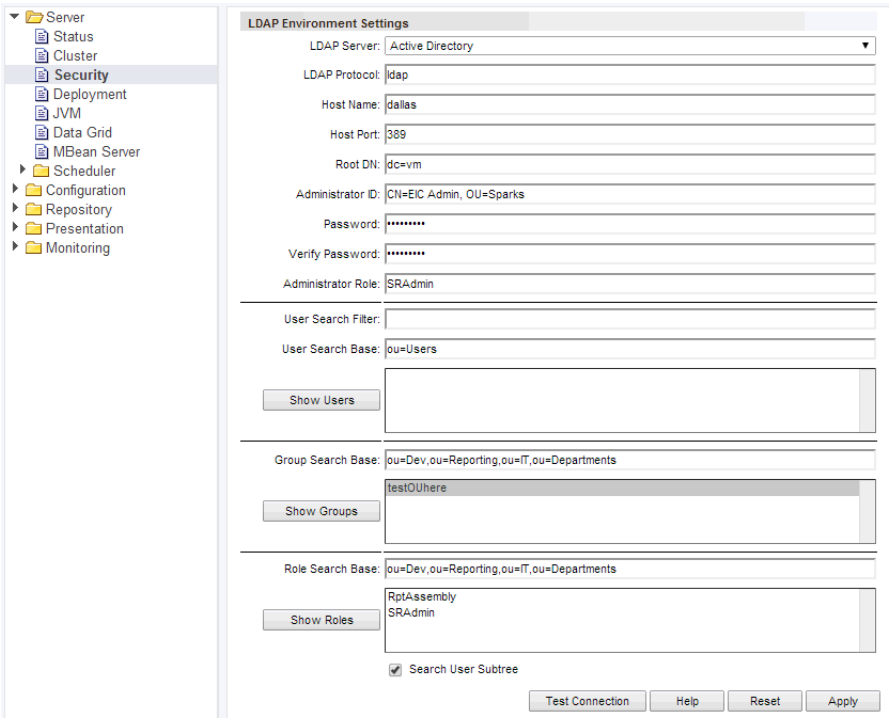
To use an Active Directory server for authentication, select ‘LDAP’ from the ‘Authentication’ menu on the ‘Security Provider’ page, and select ‘Active Directory’ from the ‘LDAP Server’ menu. (See *Specifying a Security Provider* for more information about the ‘Security Provider’ page.)

```
LDAP is recommended for authentication but not for authorization
because authorization requires modification of the LDAP schema.
```

When configuring an Active Directory server to perform authentication for Style Intelligence, take note of the following points:

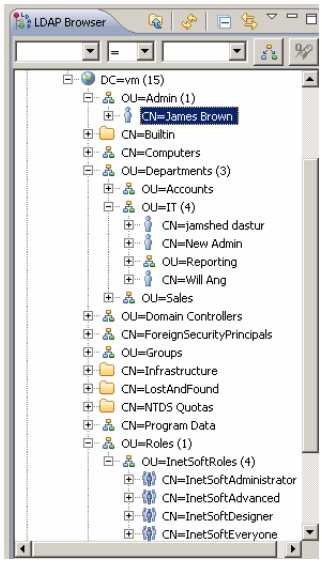
- An Active Directory schema can contain a large number of objects (users, security groups, etc). Before setting up the security provider in Enterprise Manager, use a tool like Apache Directory Studio (http://directory.apache.org/studio/downloads.html) to browse your schema and become familiar with its structure.
- It is highly recommended that you add new security groups to the Active Directory schema to support Style Intelligence users. For example, add security groups such as 'InetSoftAdmin', 'InetSoftUser', 'InetSoftDeveloper', etc., to the Active Directory schema, and then add the appropriate users to these groups.
- A *security group* in Active Directory is equivalent to a *role* in Style Intelligence.
- An *organizational unit* in Active Directory is equivalent to a *group* in Style Intelligence.
- A *search base* is the location within Active Directory from which Style Intelligence will search for and load users, security groups, etc. It is typically mapped to an organizational unit, for example, `ou=Departments`. A search base can also be a composite of multiple search bases separated by semicolons, e.g., `ou=IT,ou=Departments;ou=Sales,ou=Departments`. Press the 'Show Users', 'Show Groups', or 'Show Roles' button to verify that the entities are being read correctly.
- After you configure LDAP security, you can no longer log into Enterprise Manager using the default admin/admin credentials. You must log in as a user who has the administrator role security group.
- After you configure LDAP security, you can no longer add users/groups/roles from within the **Users** tab in Enterprise Manager. You must do this from within Active Directory. However, the **Users** tab allows you to view the users defined in Active Directory.

Figure 31. Sample LDAP Environment Setup for Active Directory

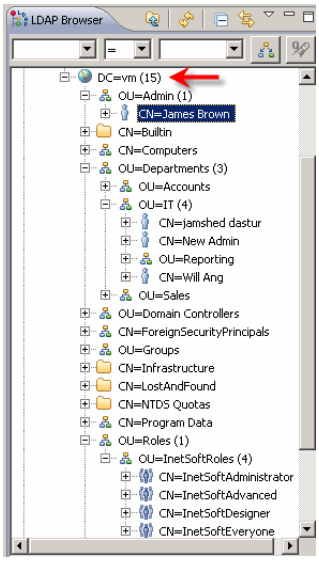


Sample Active Directory Setup

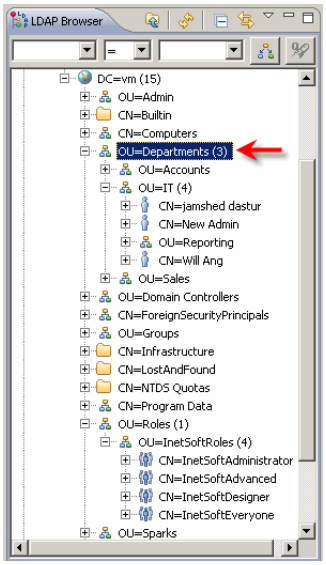
In the following example, consider the following Active Directory schema. The bullets below indicate how to fill out the 'Security Provider' page for this schema.



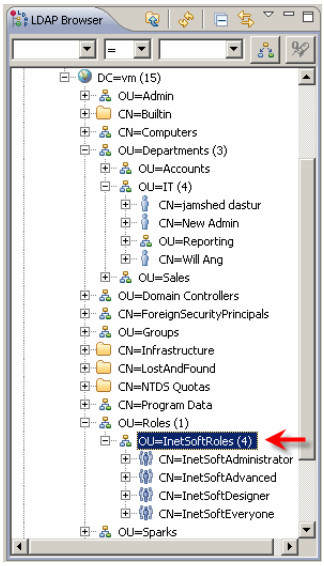
- **Root DN**: The 'Root DN' (here, "`dc=vm`") is automatically appended to all units within Style Intelligence and need not be included in any other entry.



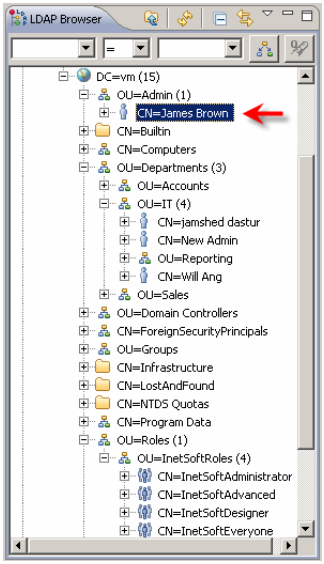
- **User Search Base/Group Search Base**: To search all units below the 'Departments' unit, set 'User Search Base' and 'Group Search Base' to "`ou=Departments`". (If you want to include only the 'Accounts' and 'Sales' units, set "`ou=Accounts,ou=Departments;ou=Sales,ou=Departments`".)



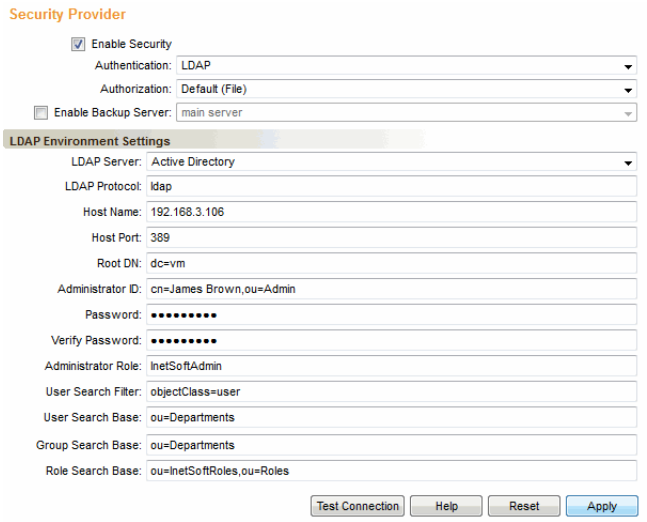
- **Role Search Base**: To search the 'InetSoftRoles' unit, set 'Role Search Base' to "`ou=InetSoftRoles,ou=Roles`".



- **Administrator ID**: The 'Administrator ID' in this case is "`ou=James Brown,ou=Admin`". Any account with sufficient read permissions can be used (preferably a service account). It does not need to be an Active Directory domain administrator, and does not need to be in the user search base.



The 'Security Provider' page for these setting appears as follows:



Writing to the Active Directory Schema

Note: It is recommended that you use the built-in security provider for authorization. See *Specifying a Security Provider*.

To use an Active Directory server for authorization, you should enable writing to the schema. To do this, follow the steps below:

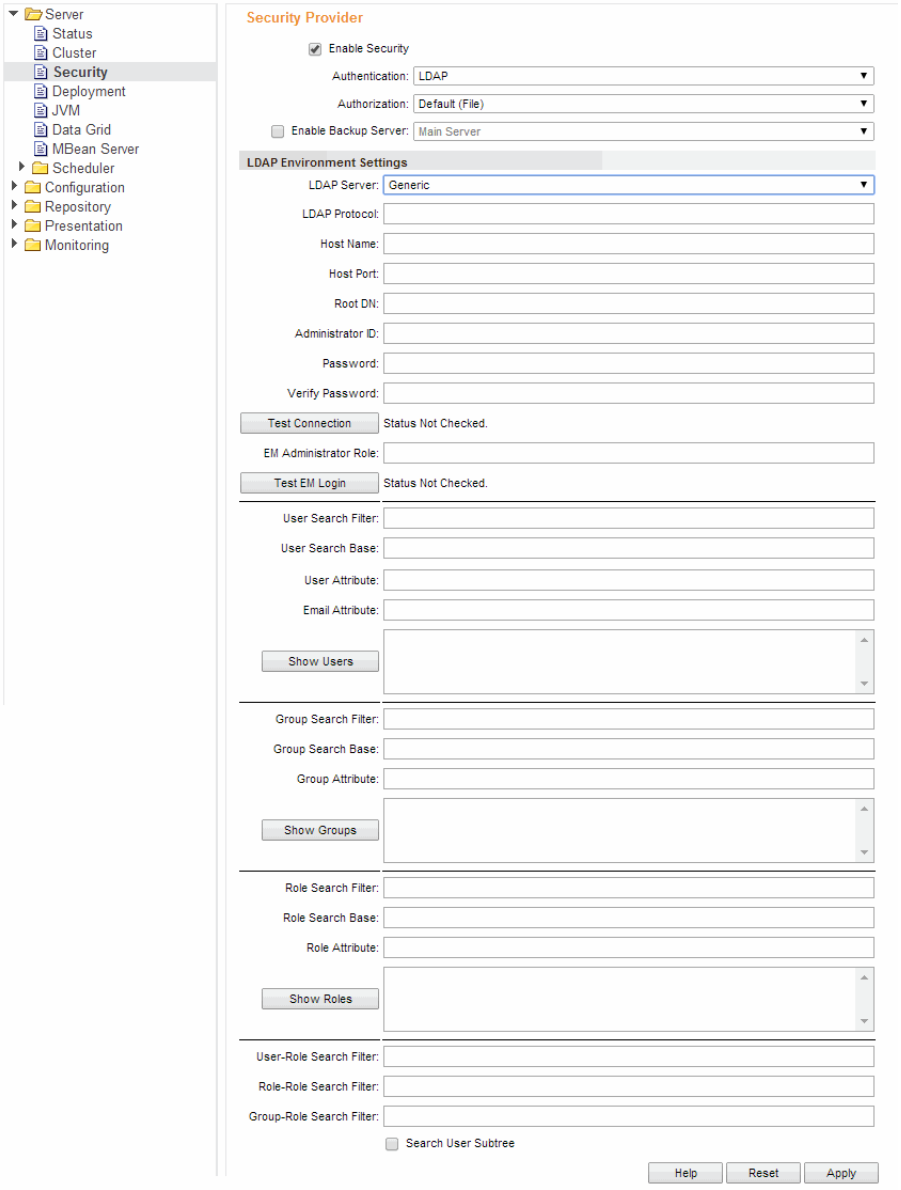
1. 'Start' >'Run'
2. Type "mmc" and click 'OK'
3. Select the 'Console' > 'Add/Remove Span-In' menu and click 'Add'.
4. Select 'Active Directory Schema' and click 'Add'
5. Select 'Active Directory Users and Groups' and click 'Add'
6. Select 'Close' and then 'OK'
7. Right-click 'Active Directory Schema' in the tree on the left
8. Select 'Change Domain Controller'
9. If the Current DC is not correct, select 'Specify Name', enter the DC, and click 'OK'
10. Right click 'Active Directory Schema' and select 'Operations Master'
11. If the Current Operations Master is not the same as the DC you just entered, click the 'Change' button
12. Check the "The Schema may be modified..." box.

13. Click ‘OK’ and exit the MMC

Note that the administrator account needs to be a member of the Schema Admins group.

Generic LDAP Server

To use any other LDAP server, select ‘Generic’ from the ‘LDAP Server’ menu under the ‘Security Provider’ page. (See *Specifying a Security Provider* information about the ‘Security Provider’ page.)



Note that a fully qualified name must be entered for the Administrator ID.

Table 8. Environment Setting Properties

PROPERTY NAME	DESCRIPTION
Host Name	The host name of the server that is running the directory server. Example: `inetsoft.com`
Port	The port number on which the directory server is listening. Example: `389`
Root DN	The distinguished name [DN] of the root of the directory server. Example: `dc=inetsoft,dc=com`
Administrator ID	The distinguished name [DN] of the directory server administrator. Example: `cn=manager,dc=inetsoft,dc=com`
Administrator Password	Directory server administrator's password. Example: `secret`
Administrator Role	The name of the administrator role in the LDAP provider.
User Search Filter	The search filter used to find system users. Example: `(objectclass=person)`
User Search Base	The base directory from which user searches will be performed. Example: `ou=People`
User Attribute	The name of the attribute in the user entry that will be used as the user ID. Example: `uid`
Group Search Filter	The group search filter used to find system groups. Example: `(objectclass=organizationalunit)`
Group Search Base	The base directory from which group searches will be performed. Example: `ou=People`
Group Attribute	The name of the attribute in the group entry that will be used as the group ID. Example: `ou`
Role Search Filter	The role search filter used to find system roles. Example: `(objectclass=groupofuniquenames)`
Role Search Base	The base directory from which role searches will be performed. Example: `ou=Groups`
Role Attribute	The name of the attribute in the role entry that will be used as the role ID. Example: `cn`

User-Role Search Filter	The user-role search filter used to find the roles assigned to a specific user. This property is a standard LDAP search filter in which the string "{0}" will be replaced with the user ID.
	Example:
	`(&(objectclass=groupofuniquenames)`
	`(uniquemember=uid={0},*))`

If using OpenLDAP, there are a few additional steps.

1. Copy {Install Directory}/lib/inetsoft.schema to

   ```
   /etc/openldap/schema/
   ```

2. Edit /etc/openldap/slapd.conf to add the following line:

   ```
   include /etc/openldap/schema/inetsoft.schema
   ```

5.1.4 Custom Security Integration

Use custom security integration when your users/roles/groups are already defined in existing systems, but the Style Intelligence server has no mechanism of directly accessing this information; e.g., database, web service (SOAP).

Composite Security Provider

The recommended approach for implementing custom security is to use the composite security provider. The composite security provider modularizes security into *authentication* and *authorization* functions. Authentication deals with retrieving user/role/group information and validating login requests. Authorization deals with assigning report- and portal-related permissions.

Required Settings for Composite Security

In order to set up composite security directly add the following properties in the ‘sree.properties’ file.

```
security.provider =
   inetsoft.sree.security.CompositeSecurityProvider
security.authentication.provider={AuthenticationProvider}
security.authorization.provider={AuthorizationProvider}
```

{AuthenticationProvider} can be one of:

- inetsoft.sree.security.FileAuthenticationProvider
- inetsoft.sree.security.ldap.ADSecurityProvider
- inetsoft.sree.security.ldap.IPlanetSecurityProvider
- inetsoft.sree.security.ldap.GenericLdapSecurityProvider
- class which extends inetsoft.sree.security.AbstractAuthenticationProvider e.g., com.company.security.MyAuthorizationProvider

{AuthorizationProvider} can be one of:

- inetsoft.sree.security.FileAuthorizationProvider

- inetsoft.sree.security.ldap.ADSecurityProvider
- inetsoft.sree.security.ldap.IPlanetSecurityProvider
- inetsoft.sree.security.ldap.GenericLdapSecurityProvider
- class which extends inetsoft.sree.security.AbstractAuthorizationProvider, e.g., com.company.security.MyAuthorizationProvider

Example: Extending AbstractAuthenticationProvider

The listing below provides an illustration of Extending the AbstractAuthenticationProvider. Please refer to the API JavaDoc for more information about these classes.

Listing 1. Extending the AbstractAuthenticationProvider Class

```
public class MyAuthentication extends
inetsoft.sree.security.AbstractAuthenticationProvider {

   public boolean authenticate(String user, Object
credential){
      //validate user credentials
      //credential is object of type
      //inetsoft.sree.security.DefaultTicket
   }
   public String[] getUsers() {
      // return a list of all user logins
   }

   public String[] getUsers(String group) {
      // return a list of usernames in the given group
   }

   public String[] getIndividualUsers() {
      // return a list of usernames not belonging to a
group
   }

   public User getUser(String name) {
      // return a inetsoft.sree.security.User object
   }

   public String[] getRoles() {
      // return a list of all Roles
   }

   public String[] getRoles(String user) {
      // return a list of all Roles for a given user
   }

   public Role getRole(String name) {
      // return a inetsoft.sree.security.Role object
   }

   public String[] getGroups() {
      // return a list of all Groups
   }

   public Group getGroup(String name) {
      // return a inetsoft.sree.security.Group object
```

```
   }

   public void tearDown() {
      //teardown the security provider
   }

}
```

```
Note: Do not override the 'findIdentity()' method of the
'AbstractAuthenticationProvider' as it is already complete.
```

It is advisable to implement internal caching within your security provider (especially if it is DB-based), because many methods such as '`getUsers()`', '`getUser()`', etc., are invoked repeatedly. Retrieval of this information from the DB for each invocation could cause performance/resource deterioration. However, if your security provider extends *AbstractSecurityProvider*, simply set '`security.cache=true`' in the `sree.properties` file to enable caching.

Example: Extending AbstractAuthorizationProvider

The listing below provides an illustration of Extending the AbstractAuthorizationProvider. Please refer to the API JavaDoc for more information about these classes.

Listing 2. Extending the AbstractAuthorizationProvider Class

```
public class MyAuthorization extends
inetsoft.sree.security.AbstractAuthorizationProvider {

   public void setPermission(String resource,Permission
perm) {
      // save the permission for a resource
   }

   public Permission getPermission(String resource) {
      // get the permission for a resource
   }

   public void removePermission(String resource) {
      // remove the permission for a resource
   }

   public void tearDown() {
      //teardown the security provider
   }

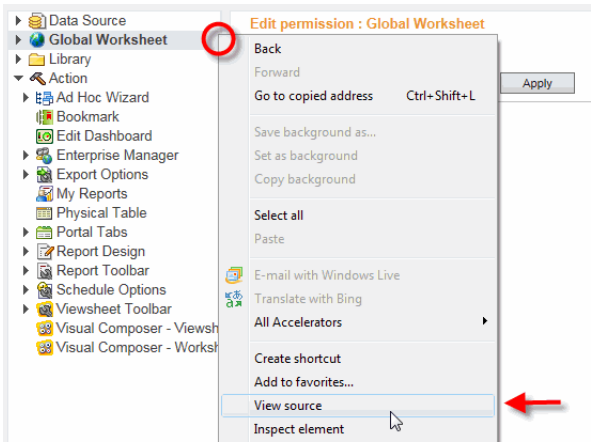
}
```

```
Note: In most cases using inetsoft's 'FileAuthorizationProvider' is
a simpler and cleaner solution. Implement your own Authorization
Provider only if you wish to store your access control information
in a store other than a file, such as a DB, etc.
```

The resource names (for which you get and set permissions) are passed using the following syntax:

RESOURCE TYPE	**RESOURCE NAME**
Replet	Name used to designate the report in Enterprise Manager or during the Ad-Hoc report generation process
Repository Folder	folder name
Report in Archive	“_archive_” + report name
Folder in Archive	“_archive_” + folder name
Query	“_query_” + query_id
Scheduled Task	“_task_” + task_id
Scheduler	“__inetsoft_schedule”
‘My Reports’ Folder	“__inetsoft_myreports”
Dashboard Design	“__inetsoft_dashboard”
Viewsheet Design	“__inetsoft_viewsheet”
Worksheet Design	“__inetsoft_worksheet”
Dashboard Portal Tab	“__inetsoft_view_dashboard”
Schedule Portal Tab	“__inetsoft_schedule”
Design Portal Tab	“__inetsoft_PortalTabs_Design”
Report Portal Tab	“__inetsoft_PortalTabs_Report”

For a current list of resource names, view the HTML source for the **Objects** tab in Enterprise Manager.



Note: Enterprise Manager authentication is integrated with Report Server authentication. When you set up an authentication provider other than the FileSecurityProvider, you can no longer log into the Enterprise Manager

using the default ‘admin/admin’ credentials. You must log in as a user which has the ‘Administrator’ role. If there is no role named ‘Administrator’ in your system, you can change it to another role by adding the ‘`role.administrator`’ property in the ‘`sree.properties`’ file. e.g., role.administrator=SRAdmin.

Porting Custom Security From v8.0 and Lower

If you already have a custom security implementation in versions 8.0 and below; the recommended approach is to re-implement your security using the new Composite security provider as described above. However if you have an existing implementation which implements the ‘SecurityProvider’ interface you may use it with the addition of some methods.

Listing 3. SecurityProvider Implementation

```
public class MyProvider implements
inetsoft.sree.security.SecurityProvider {

   public boolean checkPermission(Principal user, String
resource, char acc){
      //check the type of permission, which a user
      //has on a resource
      //user is of type SRPrincipal
      //acc is one of ‘r’,‘w’,‘d’.
   }

   public boolean supportGroupPermission() {
      //check if provider has support for setting
      //permissions on groups
   }

   public boolean authenticate(String user, Object
credential){
      //validate user credentials
      //credential is object of type
      //inetsoft.sree.security.DefaultTicket
   }
   public String[] getUsers() {
      // return a list of all user logins
   }

   public String[] getIndividualUsers() {
      //return a list of users not belonging to a group
   }
   public User getUsers(String name) {
      //return inetsoft.sree.security.User object
   }

   public String[] getRoles() {
      // return a list of all Roles
   }

   public String[] getRoles(String user) {
      // return a list of all Roles for a given user
   }

   public String[] getGroups() {
      // return a list of all Groups
   }
```

```
    public Group getGroup(String name) {
       // Get a group by name
    }

    public Identity findIdentity(Indentity identity) {
       //find the concrete identity of the security provider
    }

    public void setPermission(String resource,Permission
perm) {
       // save the permission for a resource
    }

    public Permission getPermission(String resource) {
       // get the permission for a resource
    }

    public void removePermission(String resource) {
       // remove the permission for a resource
    }

    public void tearDown() {
       //teardown the security provider
    }
}
```

5.2 Users, Roles, and Groups

Users, roles, and groups can be managed using the functions available under the **Users** tab. The navigation tree on the page under this tab includes two nodes: ‘Users’ and ‘Roles’. The ‘Users’ node lists all of the users and groups in the system and the ‘Roles’ node lists all of the roles.

Note: Users, groups, and roles can be created and configured in the Enterprise Manager only when using the ‘Default Security Provider’. While using the ‘LDAP Security Provider’, users/groups/roles will be imported from the LDAP schema and when using the ‘User Defined Security Provider’, they will be retrieved from the custom implementation.

For efficiency, the number of users listed in the navigation tree under LDAP security is limited by the em.user.max property (default 1000). Unlisted users can be found by searching. See *Searching the User/Role Tree*.

A ‘group’ usually refers to a collection of people (e.g., a project team, the sales department, etc). Users belonging to the same group may have different functions or ‘roles’ (e.g., Manager, Developer, Analyst, Tester, etc). Groups are generally organizational in nature, while roles are functional in nature.

For information on how to create users, roles, and groups, see *Defining Users, Roles, and Groups*.

5.2.1 Rules Governing Users and Roles

The following are some rules which govern users, roles and groups:

```
Note: Names of users, groups, and roles may not contain certain
characters (e.g., “&,” “/”).
```

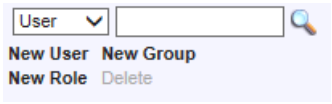
- Groups and Roles are independent, i.e., you do not have to define both groups and roles in your system. You can have just one or the other, or both, defined.
- A user can belong to more than one group.
- A user can have multiple roles
- An entire group can be assigned one or more roles.
- A role can be applied to one or more groups.
- A role can inherit all of the permissions from another role.
- Permissions to reports and other components can be assigned directly to users, groups, and roles.

5.2.2 Searching the User/Role Tree

To find specific users, roles, or groups, follow the steps below:

1. From the menu below the navigation tree, select the type of object you want to find, i.e., User, Group, or Role.
2. In the text box, enter the name of the desired user, group, or role.
3. Click the ‘Find’ button.
4. To return to the full list, clear the text box and click the ‘Find’ button.

Figure 32. Searching the User/Role Navigation tree



5.2.3 Emulating a User Login

When troubleshooting user problems, it is often useful for the administrator to log into the Portal as a particular user. This allows the administrator to experience the application environment as the user experiences it, which makes it easier to replicate the user’s problem.

This method does **not** require knowledge of the user’s password.

To login as a particular user, follow the steps below:

1. Set the following property in the *sree.properties* file:

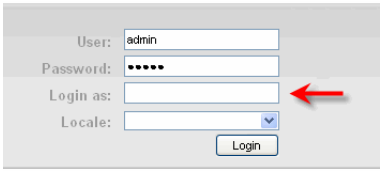
```
login.loginAs=on
```

2. In Enterprise Manager, on the 'Server' > 'Status' page (under the **Server** tab), press the 'Restart' button. This reloads the configuration settings from *sree.properties*.

3. Open the User Portal in a browser.

4. On the login screen, enter your *administrator* name and password.

   When you enter a valid administrator name (and have `login.loginAs=on`), the login screen displays a special 'Login as' field.



5. In the 'Login as' field, enter the name of the user whose environment you wish to access.

6. Click the 'Login' button.

This will effectively log you in as the user without requiring the user's password. Operations that you perform while logged in using this method (i.e., as administrator emulating a user) are listed in audit records as administrator operations, rather than as user operations.

5.3 Setting Permissions

Permissions can be set not only at a report level but also at a component level (Scheduler, dashboard, Report Explorer, etc.) and at an object level (data models, queries, worksheet assets). Report permissions are set via the **Report** tab; all other permissions are set via the **Objects** tab.

5.3.1 Repository Permissions

To set permissions for reports, dashboards, and repository folders, select the **Report** tab and expand the 'Repository' node on the tree. The 'Repository' is a collection of reports and dashboards structured into one or more folders. The repository always has a root folder: '/'. All reports, dashboards, and folders are located within this root folder. (Remember: Folders displayed in the repository are *logical* groupings, and do not represent actual directories in the disk file system.)

For instructions on how to set permissions, see *Setting Repository Permissions*.

Notes on Report Permissions

- The ‘Use Parent Permissions’ option for a report or a folder implies that it will inherit all of the permissions from its parent folder. If there are no permissions set for its immediate parent, its permission inheritance will recurse higher up in the hierarchy of folders, until the root folder is reached.
- Write permissions on a folder indicate that a user/role/group can save reports in the folder.
- Every user has full r/w/d permissions to his/her ‘My Reports’ folder.
- If a user has ‘write’ permissions on a report he/she can save over the original report template.
- If a `.srt` file is set to be ‘Read Only’ within the file system, a user cannot save over this report template.
- If a user has ‘read’ permissions (but no ‘write’ permission) on a report, he/she may save a copy of the report (‘Save As’) into any folder to which they have write permissions.
- If a user has read permissions on a report, but no read permissions on its parent folder, the report will not be visible in the repository tree. However, it can be viewed as a hyperlinked report.
- If explicitly set, the permissions of a report or a folder will override its parent permissions.

5.3.2 Component/Object Permissions

Permissions for most objects and components are set via the **Objects** tab. (See *Objects Tab* for a list of available permissions).

See *Setting Permissions* for instructions on how to do this.

Notes on Object Permissions:

- The default setting for many components is ‘Use Parent Permissions’. For components under the ‘Data Source’ node, this setting implies that no permission is granted.
- Changing permission for a data source folder has the effect of changing permissions for all data sources contained in that folder.
- Permission for the ‘My Reports’ folder is granted by default for all users. However, if explicit permission for ‘My Reports’ is assigned to at least one user, then explicit permission must be assigned for any other users who require this feature.
- The ‘Use Parent Permissions’ setting is significant only for the asset components listed under the ‘Worksheet’ node. If you set permissions for the ‘Global’ node, they can be inherited by all of the assets under this node.

See Also

Creating a New JDBC Data Source, in *Advanced Data Modeling*.
Independent Query, in *Advanced Data Modeling*.
Data Models, in *Advanced Data Modeling*.
Adding Additional Connections, in *Advanced Data Modeling*.
Creating a New Data Worksheet, in *Advanced Data Mashup*.

5.3.3 Parent Permissions

In many cases, the security settings for a given resource allow the resource to inherit permissions from its "parent". To enable inheritance when available, select 'Use parent permissions'.



The following table lists the "parent" for each type of resource capable of inheriting parent permissions:

RESOURCE TYPE	PARENT
Report	The repository folder containing the report on the **Report** tab tree.
Viewsheet	The repository folder containing the Viewsheet on the **Report** tab tree.
Folder	The repository folder containing the folder on the **Report** tab tree.
Datasource	The root 'Data Source' node on the **Object** tab tree.
Connection	The same parent as the datasource on which the connection is defined.
Query	The datasource on which the query is defined.
Data model	The datasource on which the data model is defined.
Data Worksheet	The 'Global Worksheet' node on the **Object** tab tree.
Library asset	The 'Library' node on the **Object** tab tree.

See Also

Repository Permissions, to set permissions for reports, Viewsheets, or repository folders.
Component/Object Permissions, to set permissions for all other resources (queries, data sources, Data Worksheets, etc.) and Portal actions.

6 Scheduler

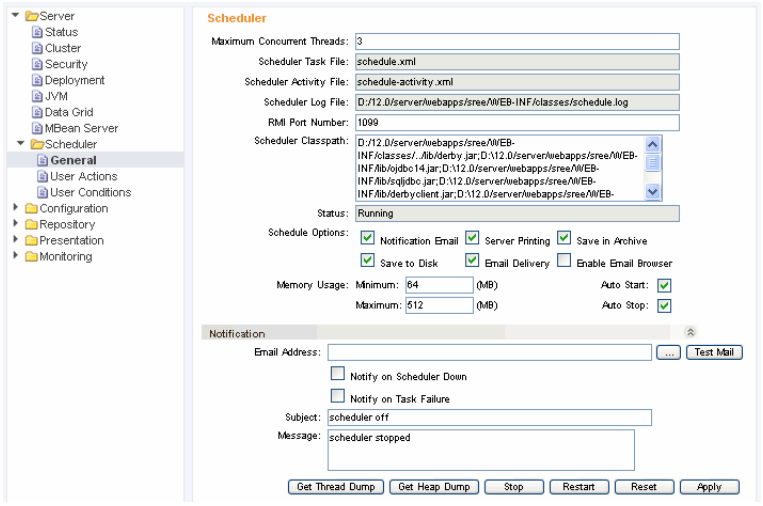
The ‘Scheduler’ is an application that facilitates scheduled batch report and Viewsheet generation. An administrator or an end user (with scheduler permissions) can access this tool via the Enterprise Manager or the User Portal respectively. You can create one or more scheduled ‘tasks’ for which you specify a time condition, reports or Viewsheets to be generated, delivery mechanism (email, print, save to disk, archive, etc.), and format (PDF, Excel, etc.) of the generated output.

The ‘Scheduler’ is a stand-alone application. It runs independently of the report application server (Tomcat, WebLogic, WebSphere, etc.) as a separate process. Therefore, even if the report server goes down, the Scheduler continues to run. The Scheduler runs as an RMI server and can also be accessed programmatically. This is discussed in greater detail in the *RMI Server* section.

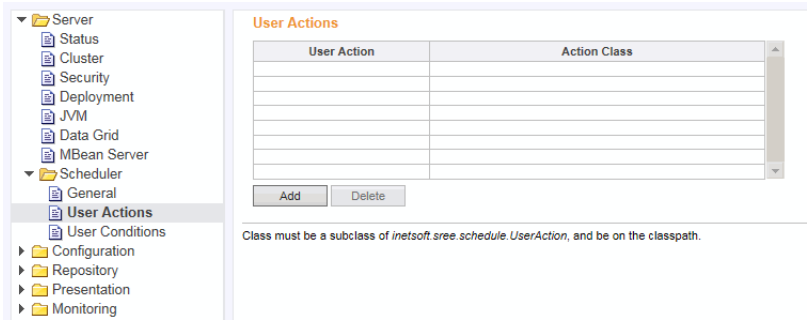
6.1 Configuring the Scheduler

The ‘Scheduler’ can be configured using the Enterprise Manager. Click the **Server** tab and select the ‘Server’ > ‘Scheduler’ > ‘General’ page on the navigation tree.

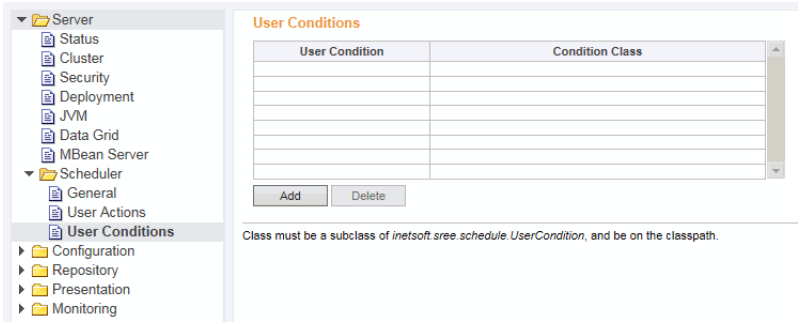
Figure 33. Scheduler Configuration Page



The ‘Server’ > ‘Scheduler’ > ‘User Actions’ page allows you to create your own Scheduler Action, which will be available when the ‘User Defined’ option is selected from the Scheduler’s **Action** tab (in Enterprise Manager or the User Portal). See *Scheduler Actions* for more information about implementing an action.



The ‘Server’ > ‘Scheduler’ > ‘User Conditions’ page allows you to create your own Scheduler Condition, which will be available when the ‘User Defined’ option is selected from the Scheduler’s **Condition** tab (in Enterprise Manager or the User Portal). See *Scheduler Conditions* for more information about implementing a condition.



6.2 Scheduler Properties

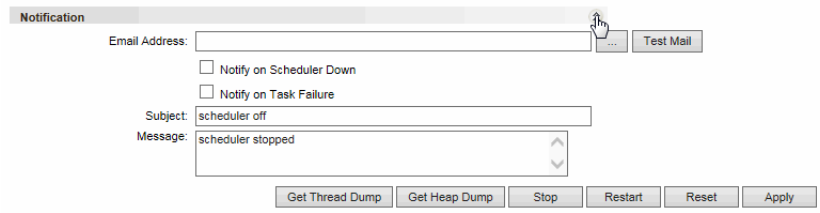
The table below lists the Scheduler properties.

Table 9. Scheduler Properties

Maximum Concurrent Threads	Set the maximum number of threads that can be executed concurrently.
Schedule Task File	The task file (*schedule.xml*) is stored in the sree.home directory. This file contains the definition information for tasks, and their conditions and actions.
Schedule Activity File	The activity file (*schedule-activity.xml*) is generated dynamically when tasks are created. It is stored in the *sree.home* directory. The activity file is used to maintain the status of tasks and actions, and is used to implement rollback/recovery if the Scheduler should fail.
Scheduler Log File	Contains log messages generated by the Scheduler
RMI Port Number	The Scheduler server is an RMI application. By default RMI uses port 1099. If the default RMI port is not used, it can be changed here.

Scheduler Classpath	The classpath that will be prepended to the system classpath when the Scheduler process is started. The scheduler is a separate process with a separate JVM.
Status	Shows whether Scheduler is running or stopped.
Auto Start	Allows the scheduler to be started automatically on initialization of repository servlet. This option does not apply for the cluster environment.
Auto Stop	Allows the scheduler to be stopped automatically when the repository servlet terminates. This option does not apply for the cluster environment.
Schedule Options	Select options available to a user when they schedule tasks, such as Notification Email, Save In Archive, etc.
Memory Usage	Sets the minimum and maximum memory usage for the Scheduler.
Stop	Stop the Scheduler application.
Restart	Restart the Scheduler application.
Get Thread Dump	Download a text file containing a thread dump of the Scheduler process.
Get Heap Dump	Download an HPROF file containing a heap dump of the Scheduler process.

To configure notifications for Scheduler status, expand the 'Notification' panel at the bottom of the page. This allows the server to send a one-time message if a Scheduled task or the Scheduler application stops abnormally. A 'Mail Host' and 'From Email Address' must be specified on the 'Mail Configuration' page (see *Configuring Server Email*).



The following options are available:

Email Address	A comma- or semicolon-separated list of email addresses to which Scheduler notifications should be sent.
Notify When Scheduler is Down	Send a notification if the Scheduler application stops abnormally.
Subject	Subject line of the email.
Message	Body of the email.
Notify When Task Fails	Send a notification if a scheduled task could not be completed.

6.2.1 Notes on Scheduler Configuration

- Use the 'Restart' and 'Stop' to restart and stop the Scheduler. The RMI registry will be started automatically. (These options do not apply for the cluster environment. See *Configuring Server Clustering*.)
- If you must specify additional parameters to the Java executable, add a `schedule.java.opts` line to the *sree.properties* file.
- To invoke a custom listener class at the start and end of each scheduled task, specify the fully-qualified name of the class as the value of the `schedule.task.listener` property in the *sree.properties* file.
- Since the Scheduler runs in its own Java Runtime Environment, you must specify the product's JAR files in the 'Scheduler Classpath'.
- For WebSphere 3.5, add the *etools.jar* and product JAR files (*sree.jar*, *visual.jar*, or *bisuite.jar*) to the `com.ibm.ejs.sm.adminServer.classpath` property in the `WS_HOME/appserver/bin/admin.config` file.

6.3 Scheduler Tasks

A scheduler task defines the batch operation to be performed. It specifies the time condition, the reports or Viewsheets to be generated and their delivery mechanism (email, print, archive, save to disk, etc.). Both the Enterprise Manager and the User Portal provide interfaces to create, edit, and monitor scheduled tasks.

See Also
Configuring the Scheduler, for information on setting scheduler properties.

6.3.1 Creating a Scheduler Task

To create a scheduled task, follow the steps below:

1. Select the **Schedule** tab and click on the 'New Task' link, located below the navigation tree. This creates a new task, with the default name 'Task n' (e.g., Task1, Task2, Task3, etc.).

2. Select the **Condition** tab and specify a condition to determine when the task will execute. See *Scheduler Conditions*.

3. Select the **Action** tab and specify the type of action to take when the scheduled task executes. See *Scheduler Actions*.

4. Select the **Options** tab and set options for task execution. See *Scheduler Options*.

6.3.2 Scheduler Conditions

A scheduler condition is used to determine when the scheduled task executes. There are three types of conditions: Time conditions, Chained (Completion) conditions, and User Defined conditions.

- **Daily**: This task will execute every n days (1 day, 2 days, 3 days, etc.) at a specified time. The time is specified in HH:mm [am|pm], e.g., 11.39 pm. You can also select the weekday option which will execute the task at the specified time on weekdays.
- **Weekly**: This task will execute every n weeks (1 week, 2 weeks, 3 weeks, etc.) at a specified time on a specified day of the week or, optionally, every day of the week. The time is specified in HH:mm [am|pm], e.g., 11.39 pm.
- **Monthly**: This task will execute every nth day of the specified month/s, or every month. (e.g., 21st day of February, 25th day of November, etc.) at a specified time; or every nth day of the week of a month/s, or every month (e.g., 2nd Sunday of April), at a specified time. The time is specified in HH:mm [am|pm], e.g., 11.39 pm.
- **Run Once**: This task will execute once on a certain day at a specified time, e.g., Nov 25, 2006 at 11.30 am. The time is specified in HH:mm [am|pm], e.g., 11.39 pm.
- **Chained Condition**: A chained condition is based on the completion status of another scheduled task or a scheduled cycle. The condition evaluates to true only when the specified task or the cycle completes successfully, and not otherwise. A completion condition can be specified by selecting the ‘Chained’ option and then specifying the scheduled task or the scheduled cycle.
- **User Defined Condition**: In addition to the Time and Chained conditions, User Defined conditions can also be used for a scheduled task.

In order to use a User Defined Condition, do the following:

1. Create a class which implements

   `inetsoft.sree.schedule.UserCondition`

   or extends

   `inetsoft.sree.schedule.DefaultUserCondition.`

2. Place this class on the CLASSPATH of the Scheduler.

3. Select ‘User Defined’ as the condition type in your task.

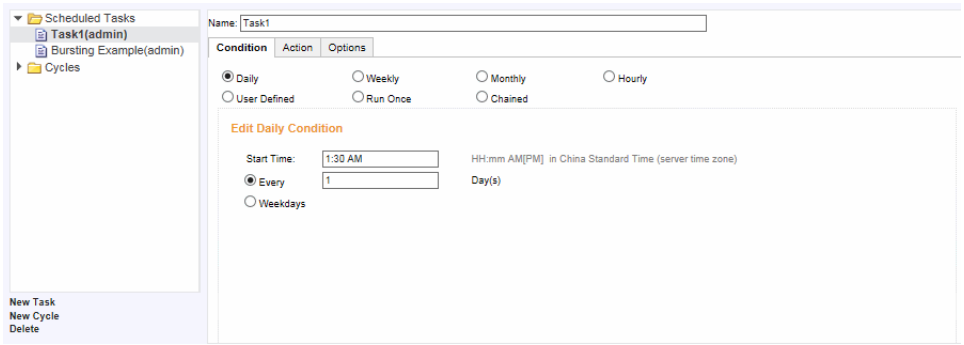
4. Specify the condition's fully qualified class name in the **Condition** tab of the task definition.

For ease of use, you can add a line to *sree.properties* that sets `replet.viewer.conditions` to a comma-separated list of classes. This will populate a drop-down that will set the correct class. When selected, the parameter table will also be populated with the available parameters and default values. See *Configuring the Scheduler* to register a reusable user-defined condition.

6.3.3 Specifying Multiple Conditions

It is possible to specify multiple conditions. Click on the 'Multiple Schedules' button at the bottom right of the editing panel. This will display the schedule condition list, to which you can add, delete, or edit conditions by clicking on the 'Add', 'Delete', and 'Edit' buttons respectively.

Figure 34. New Scheduler Task



6.3.4 Scheduler Actions

The scheduler action specifies the operations to be performed when the scheduled task runs. There are four types of actions: Report Action, Viewsheet Action, Burst Action, and User Defined Action.

Report Action

A report action executes one or more reports and delivers them as emails, saves them in the report archive, prints them via a server printer, and/or saves them to disk.

Report Delivery Options

The following table explains the available delivery options.

Table 10. Delivery Options

OPTION	DESCRIPTION
Save in Archive	Save the generated report in the report archive. A report archive must be properly configured, and the scheduler user must have proper permissions to save the report in the selected folder. The filename can include parameters for automatic insertion of report name and time/date of report generation. See the *Filename Parameters* section below for examples.
Print on Server	Print to one or more printers. The printer name must be accessible from the host machine. The printing mechanism is done through the custom driver (the JDK default printing does not support selecting a printer by name), and is under the same constraints as the custom drivers.
Deliver to Emails	Send a report to other users via email. Press the '...' button next to the 'To' field to open the 'Select Emails' dialog box and choose emails from a list. Enter an email in the 'Notify on Failure' field to send an email in the event the scheduled task fails to execute. To embed the report within the email as HTML, select the 'HTML Email' option. (Note that the appearance of the delivered report may not be fully preserved if the recipient's email client is not standards compliant). To send the report as an attachment, select one of the other formats (PDF, Excel, etc.). If the file is too large, it will be split into multiple emails. The maximum size allowed for each email can be set from the 'Configuration' > 'Mail' page under the **Server** tab. The email subject line can include parameters for automatic insertion of report alias and time/date of report generation. The subject line specified here overrides the setting in 'Configuration' > 'Mail' under the **Server** tab. See *Configuring Server Email* for more information. The 'Bundled as zip' option allows you to zip the attachment and optionally specify a password to encode the archive with WinZip 256-bit AES encryption. See below for configuration requirements.
Save to Disk	Save the report on the local file system in any of the following formats: PDF, HTML, Excel, PowerPoint, RTF, SVG, CSV, Text, XML. The 'Path' field should specify a valid absolute path, including filename. (The filename extension is added automatically.) The filename can include parameters for automatic insertion of report name and time/date of report generation. See the *Filename Parameters* section below for examples.
Notify when Completed	Send email notifications to users about task completion status. Optionally you can check 'Notify only if failed'.

Email attachments that are bundled as zip files can be password protected. This feature is available by default for new installations on Windows and Mac. For Linux installations and upgrades from previous versions, follow the steps below:

1. Open the 'Server' -> 'JVM' page under the **Server** tab, and note the 'JVM Version' and 'Java Home Directory' listed there.

2. For JVM version 1.6 or 6, download the Version 6 JCE Unlimited Strength zip file. For JVM version 1.7 or 7, download the Version 7 JCE Unlimited Strength zip file.

3. Unzip the file, and place the contents (*local_policy.jar* and *US_export_policy.jar*) in the *lib/security* folder of your Java Home Directory.

4. Restart the Style Intelligence server.

See Also
Filename Parameters, for information on customizing the filename for saved reports.

Filename Parameters

The filename you specify for the 'Save to Disk' option, and the attachment for the 'Deliver to Emails' option can include parameters to automatically insert the report name and time/date of report generation.

These parameters use the *java.text.MessageFormat* syntax, where "`{0}`" represents the report alias (report name, if no alias is specified), and "`{1,date}`" represents the time/date of report generation. You can specify a format for the date parameter using the *java.text.SimpleDateFormat* syntax. Some examples of automatically generated filenames are shown below.

SPECIFIED FILENAME	GENERATED FILENAME
myRep_{1,date}.pdf	myRep_Nov 14, 2007.pdf
{0}_{1,date}.pdf	Production_Nov 14, 2007.pdf
myRep_{1,date,MMM-dd}.pdf	myRep_Nov-14.pdf
myRep_{1,date,EEE-h-mm a}.pdf	myRep_Wed-11-49 AM.pdf
myRep_{1,date,yy-MM-dd-HH-z}.pdf	myRep_07-11-14-11-EST.pdf

Because of filename restrictions, you cannot use the "`{1,time}`" syntax that is available for email subject lines (see *Configuring Server Email*).

To include a report's creation parameter values in the file name, simply place each parameter name within curly braces. The parameter names you enter in the filename must exactly match the parameter names in the 'Creation Parameters' panel of the **Action** tab. For example, if a parameter called "stateParam" is assigned a value of "NJ" in the 'Creation Parameters' panel, then the filename

```
myRep_{1,date,MMM-dd}_{stateParam}.pdf
```

will yield a generated filename of

```
myRep_Nov-14_NJ{0}.pdf
```

Viewsheet Action

A Viewsheet action executes one or more Viewsheets, and delivers them as emails or simply transmits a notification. The following table explains the available delivery options.

Table 11. Delivery Options

OPTION	DESCRIPTION
Deliver to Emails	Send a Viewsheet to other people as an email attachment. The report can be sent in PDF format, or in any of the other export formats. If the file is too large, it will be split into multiple emails. The maximum size allowed for each email is defined by the 'mail.attachment.max' property, which can be set from the 'Configuration' > 'Mail' page under the **Server** tab. The email subject line can include parameters for automatic insertion of Viewsheet alias and time/date of generation. The subject line specified here overrides the setting in 'Configuration' > 'Mail' under the **Server** tab. See the *Configuring Server Email* section for more information.
Notify when Completed	Send email notifications to users about task completion status. Optionally you can check 'Notify only if failed'.

To make a scheduled action conditional based on actual Viewsheet data, place a call to *thisViewsheet.scheduleAction(Boolean[,emails])* within the Viewsheet script. This allows you to run or suppress the delivery options based on your own custom Viewsheet logic, changing a scheduled task into a data-driven alert notification.

Burst Action

Report bursting is a batch reporting feature in which a large report is generated offline and internally segmented, or "bursted," according to users or roles. When each user views this burst report, they will see only the segment targeted to them. Using a scheduled burst action, it is possible to deliver each segment of this report to the appropriate user/role. See the *Report Bursting* section for more information.

A Burst Action can be associated with a task by choosing 'Burst' as the action type and specifying the appropriate property values.

User-Defined Action

In addition to the 'Report' and 'Burst' actions, 'User Defined' actions can also be specified for a scheduled task. Using a 'User Defined' Action involves the following steps:

- Create a class which implements ‘`inetsoft.sree.schedule.UserAction`’, or extends ‘`inetsoft.sree.schedule.DefaultUserAction`’.
- Add this class to the CLASSPATH of the scheduler.
- Select ‘User Defined’ as the action type in the task.
- Specify the action’s fully qualified class name in the **Action** tab of the task definition.

For ease of use, you can add a line to *sree.properties* that sets `replet.viewer.schedule.actions` to a comma-separated list of classes. This will populate a drop-down menu that will set the correct class. When selected, the parameter table will also be populated with the available parameters and default values. See *Configuring the Scheduler* to register a reusable user-defined action.

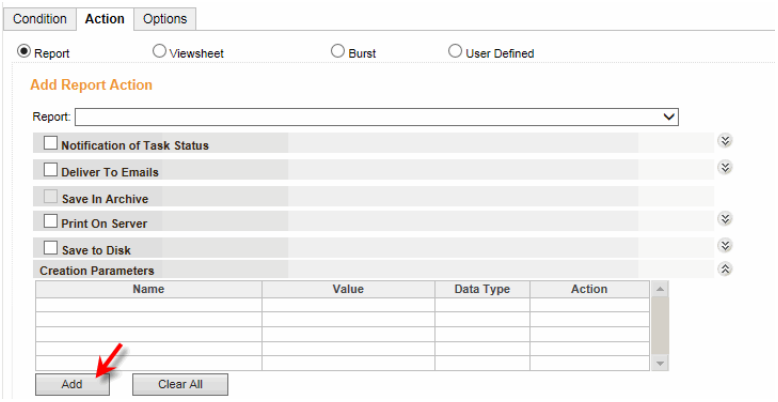
Specifying Multiple Actions

It is possible to specify multiple actions. Click on the ‘Multiple Actions’ button in the bottom right of the editing panel. This will display the scheduled action list, to which you can add, delete, or edit actions by clicking on the ‘Add’, ‘Delete’ and ‘Edit’ buttons, respectively.

Creation Parameters

To pass parameters to the scheduled report or Viewsheet, follow the steps below:

1. Press the ‘Add’ button below the ‘Creation Parameters’ table.

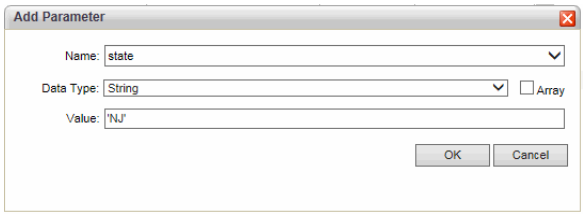


   This opens the ‘Add Parameter’ dialog box.

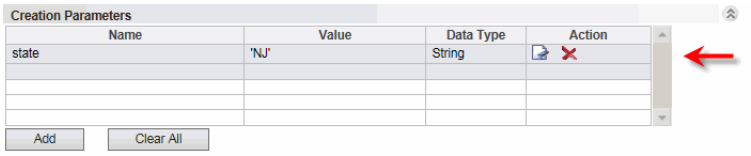
2. In the ‘Name’ field, select an existing parameter or manually enter the name of a parameter.

3. In the ‘Data Type’ field, select the data type of the parameter. Select ‘Array’ if the parameter requires an array of values.

4. In the ‘Value’ field, enter the value to be passed into the parameter. Enter a comma-separated list for an array parameter.



5. Press ‘OK’ to close the dialog box. The parameter that you added is now listed in the table.



6. (Optional) To modify the parameter settings, press the ‘Edit’ button in the table next to the parameter. To remove the parameter, press the red ‘X’ in the table next to the parameter.

7. Press the ‘Save’ button at the bottom of the screen to save the settings.

To add additional parameters, press the ‘Add’ button below the ‘Creation Parameters’ table and repeat the above steps. To remove all parameters, press the ‘Clear All’ button under the ‘Creation Parameters’ table.

See Also

Parameter Tab, to assign parameters for non-scheduled report generation.

6.3.5 Setting an Alert

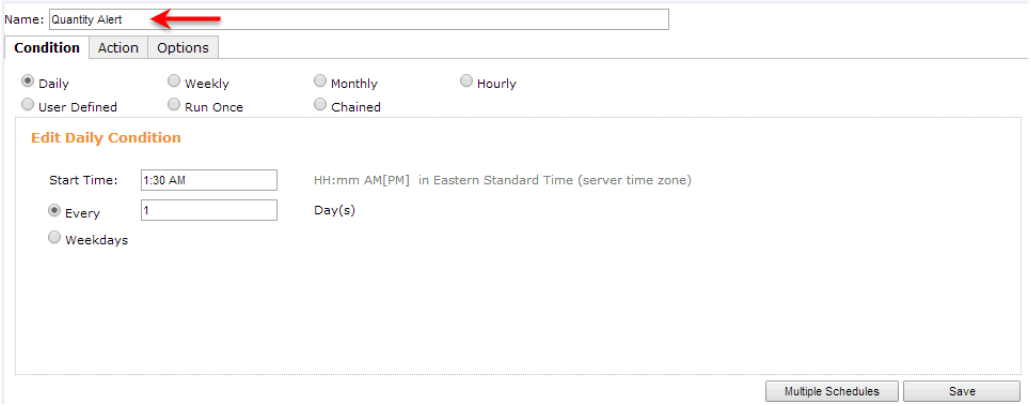
An alert is a notification that a specified event has occurred, for example, that total sales have fallen below some critical level. You can set an alert based on highlight conditions in a report or Viewsheet, or based on Viewsheet Output component ranges.

To configure an alert for a report or Viewsheet, follow the steps below:

1. Select the **Schedule** tab in the Enterprise Manager.

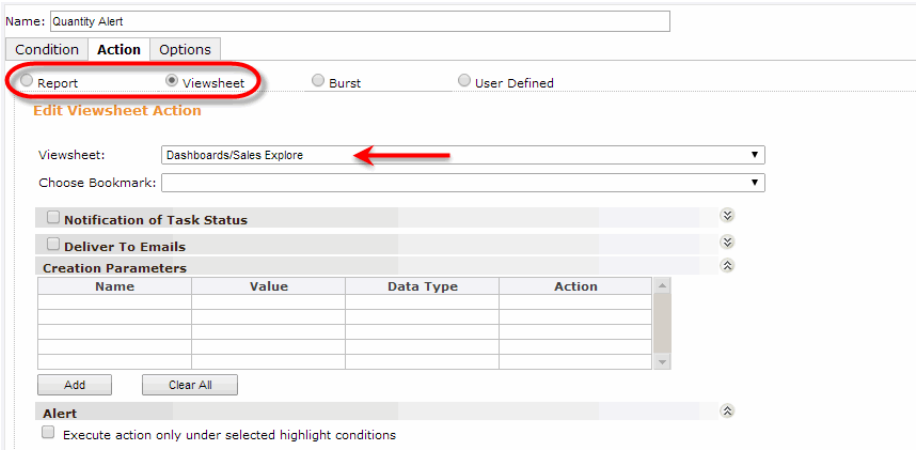
2. Press the ‘New Task’ button below the Schedule table. This creates a new task with the name ‘Task1’, and opens the Editor.

3. Change the task name as desired in the 'Name' field.



4. On the **Condition** tab, set the frequency with which the alert conditions should be tested. (See *Scheduler Conditions* for more details on the available day and time conditions.) For example, if you want to test a report highlight condition every day, specify a 'Daily' condition.

5. On the **Action** tab, specify the report or Viewsheet that contains the alert condition you want to test.

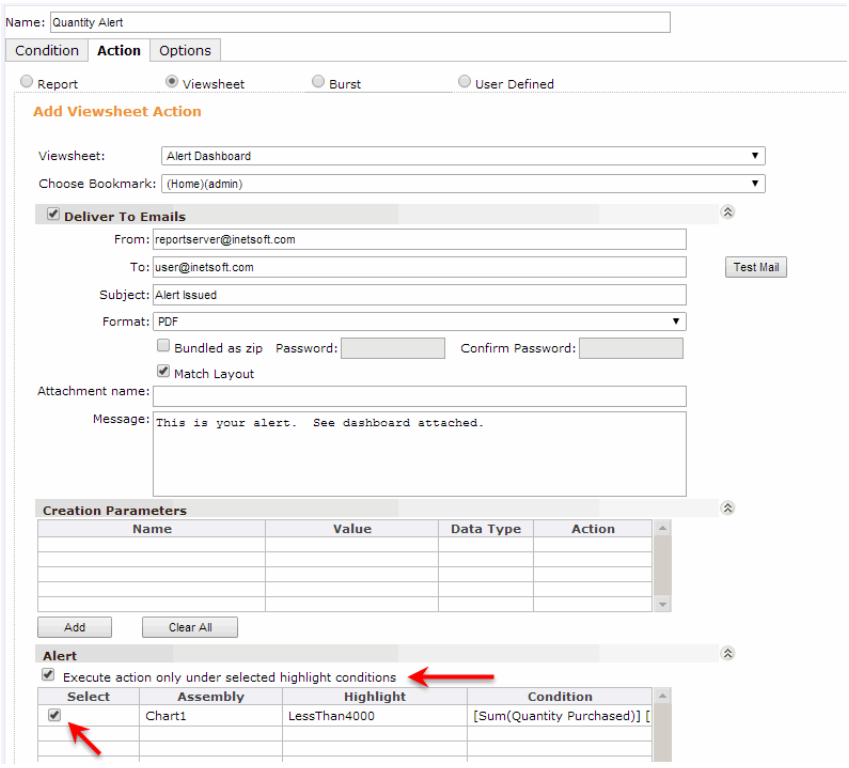


6. Select the actions that you want to perform if the alert is triggered. Generally, you will at least select the 'Deliver To Emails' option. (See *Scheduler Actions* for more about the available actions.) Enter a 'Subject' and 'Message' appropriate for the nature of the alert.

7. In the 'Alert' panel at the bottom of the **Action** tab, check the 'Execute action only under selected highlight conditions' option. This displays a list of the available highlight and range conditions for the selected Viewsheet or report.

The table lists highlight conditions for all chart, table, image, and text components, as well as range conditions for Viewsheet Output components.

8. In the table, select the highlight or range conditions that you want to trigger the alert. When *any* of the selected conditions test true, the actions you specified above (e.g., emailing) will be performed.



9. Press 'Save' to save the scheduled task.

See Also

Setting an Alert, in *Using the Portal*, for information on how an end-user can set an alert.

Highlights, in *Advanced Dashboard Design*, for information on creating Viewsheet highlights.

Highlighting, in *Advanced Report Design*, for information on creating report highlights.

Output Component Properties: Advanced Tab, in *Advanced Dashboard Design*, for information on adding ranges to Output components.

6.3.6 Scheduler Options

There are several options available when executing a scheduler task.

- **Enabled**: A task can be temporarily enabled or disabled by selecting or deselecting this option.
- **Delete if not scheduled to run again**: If a task is scheduled to run once, this option will delete it from the system once it runs.

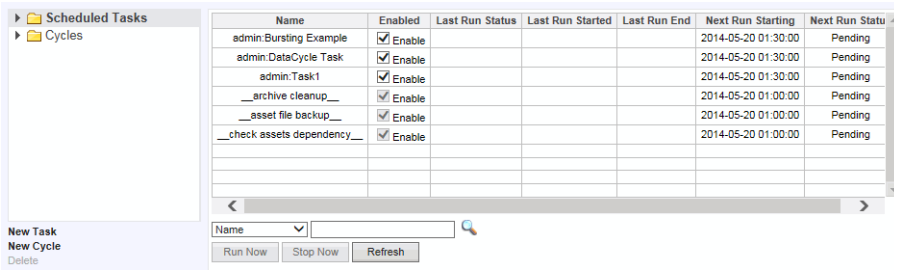
- **Start From and Stop On**: A task will only execute within the specified date range.
- **Execute As**: A task can be executed as a user, in which case user permissions will influence the execution of the task. For example, if VPMs are set up (data-level security; see *Virtual Private Model Security* in *Advanced Data Modeling*), different users will see different data. All of the existing users who have permission to use the scheduler will be included in the drop down list.
- **Locale**: A different locale can be specified for each individual task by using this option.

6.4 Monitoring and Managing Scheduler Tasks

The status of all scheduled tasks can be monitored from Enterprise Manager:

- Select the **Schedule** tab and select the 'Scheduled Task' node from the tree. A list of all of the scheduler tasks is displayed. You can view the start time, the end time, the status of the last run, and also the start time for the next scheduled run.
- Temporarily disable or enable a task by using the checkbox under the 'Enabled' column.
- Click on a task name to select it. Click the 'Run Now' button to execute the task immediately.
- Click the 'Stop Now' button to halt a task in progress.

Figure 35. Scheduled Tasks Page

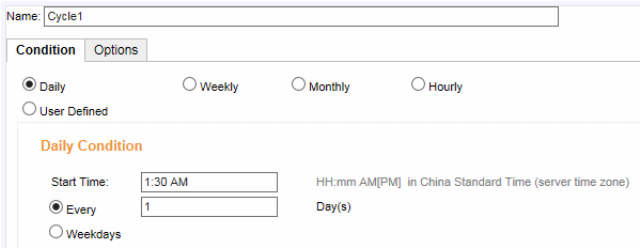


6.5 Scheduler Cycle

A scheduler cycle is a single time condition. Scheduler (data) cycles are used to specify generation times to refresh pregenerated reports, e.g., a report executes every day at 7:00 am.

A new cycle can be created by clicking on the 'New Cycle' link, located below the navigation tree. Refer to the *Scheduler Conditions* chapter for details on the different time conditions available.

Figure 36. Creating a new Scheduled Cycle



6.6 Programmatic Scheduler Access

The scheduler can be started and stopped from the Enterprise Manager interface, however, the scheduler can also be used as a standalone application through the ScheduleServer class. It has a main method so that it can be run as a process.

```
java -Dsree.home=. inetsoft.sree.schedule.ScheduleServer
```

If the classpath is not already set (e.g., by an environment variable), the classpath should also be specified, as follows:

```
java -cp {CLASSPATH} -Dsree.home=.
inetsoft.sree.schedule.ScheduleServer
```

where `{CLASSPATH}` is typically

```
{InetSoftInstallation}\server\webapps\sree\WEB-
INF\lib\bisuite.jar;
{InetSoftInstallation}\server\webapps\sree\WEB-
INF\lib\etools.jar
```

The launching application can then get an instance of the Scheduler object:

```
Scheduler scheduler = Scheduler.getScheduler();
```

The Scheduler API can be used to programmatically add tasks and perform other management functions. However, this is normally not necessary because the Enterprise Manager allows for easy visual configuration. The only exception is tasks with user-defined conditions, which have to be created programmatically and added to a scheduler from the launching application.

The scheduler can be run as a thread inside another Java process. The scheduler creates a number of internal threads when the main thread is started. The main thread can be started using the `Scheduler.start()` static method. The scheduler is a singleton object. This means there is always one scheduler instance in any JVM process. The `start()` method can be called multiple times, and the call is ignored if a scheduler has already been created.

```
Scheduler.start();
```

6.7 Automated Alerts

You can implement an *automated alert* by creating a scheduled task that executes when specified conditions are met by report or Viewsheet data. This allows you to automatically inform a user when results within the data demand their attention (e.g., inventory falls below a certain level, order volume spikes, etc.).

To create an automated alert, follow the steps below:

1. Design the scheduled task as described in *Creating a Scheduler Task*. Take note of the following:

    a. In the 'Report' or 'Viewsheet' menu of the **Action** tab, choose the report or Viewsheet which contains the data you wish to evaluate.

    b. Configure the task actions (notification, email, etc.) to accomplish the desired "alert" behavior. For example, if you wish to generate an email to users when a Viewsheet data indicator meets certain criteria, configure the 'Deliver to Emails' message to convey the desired alert.

    c. Configure the settings on the **Condition** tab to execute the task at the interval at which you wish to evaluate the report or Viewsheet data.

2. Within the specified report or Viewsheet, implement the logic which determines whether an alert should be dispatched. In general, you should place the script containing this logic on the individual component containing the data to be evaluated.

3. Within the report or Viewsheet script, place a call to `scheduleAction()` to specify whether the scheduled task should execute or not (based on the logic you implemented above).

    a. For a report, *scheduleAction(Boolean[,emails])* is a property of individual report elements. If a report element sets `scheduleAction(false)`, the scheduled task will not execute.

    b. For a Viewsheet, *thisViewsheet.scheduleAction(Boolean[,emails])* is a property of the Viewsheet itself. If a Viewsheet script sets `thisViewsheet.scheduleAction(false)`, the scheduled task will not execute.

4. Deploy the report or Viewsheet to the repository as you usually would.

When the scheduled task executes (as determined by the Scheduler settings on the **Condition** tab), the Scheduler will check to see whether `scheduleAction()` is set to `true` or `false` for the report or Viewsheet. If the value is `false`, the Scheduler aborts the task actions, and no alerts are issued. If the value is `true`, the Scheduler proceeds to execute the task actions, and any specified alerts are delivered to users.

See Also

scheduleAction(Boolean[,emails]), for more information about using the report property.

thisViewsheet.scheduleAction(Boolean[,emails]), for more information about using the Viewsheet property.

7 Presentation

This section discusses the various features that are available for tailoring the appearance of the User Portal.

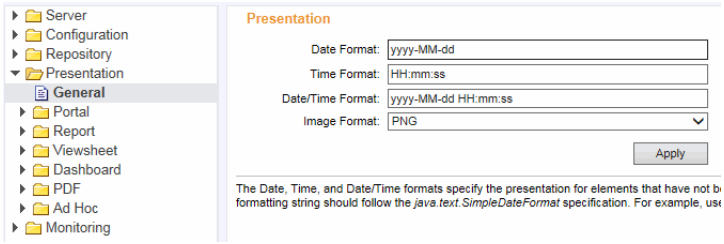
7.1 General Presentation

The ‘Presentation’ > ‘General’ page allows the Repository to be presented in a sorted format by specifying a sorting order. If ‘Sort Repository Tree’ is set to none, the user can order its replets and the order will be obeyed by the Portal.

You can also configure default Date and Time formats. (See *Formatting* in the *Advanced Report Design* for more information about formats.)

The ‘Image Format’ option determines the encoding format for graphics, such as painters (charts) displayed in the browser. PNG images have better quality and a smaller file size than JPEG images, but PNG images are only supported by version 4 or higher browsers.

Figure 37. The ‘Presentation’ > ‘General’ Page



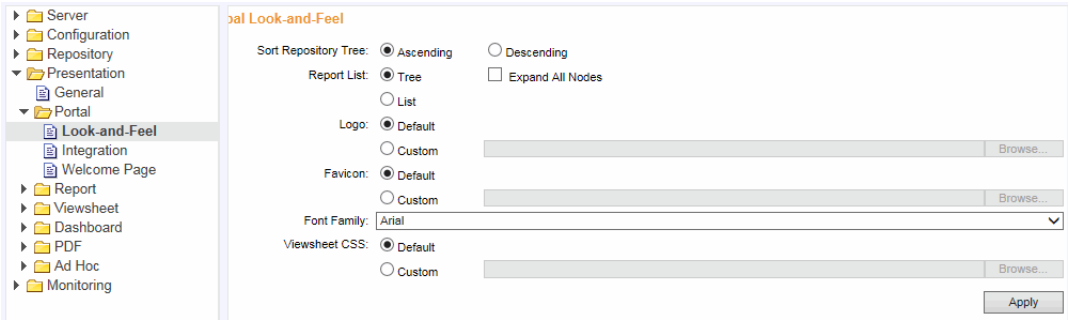
Under the ‘Presentation’ node are options to configure the appearance of Portal items, presentation of web-based reports, and PDF export options, respectively.

7.2 Portal

To change the appearance of the User Portal, use the settings available on the pages under the ‘Presentation’ > ‘Portal’ node on the navigation tree.

7.2.1 Look-and-Feel

Set the global look-and-feel for the entire Portal by adjusting the properties shown below. You can also control the look-and-feel of the Portal page via a CSS file. See *Portal CSS Customization* for more details.



- **Sort Repository Tree**: Specifies the sort order of items in the Repository listing.
- **Report List**: Controls the Portal's Repository style. The default is 'Tree', which displays Repository folders and reports in an hierarchical structure. The 'Expand All Nodes' option pre-expands all of the folders in the Repository tree. The 'List' option displays Repository folders and reports as a flat list using the entire page.
- **Logo**: Specifies the top-left logo in the User Portal and Enterprise Manager. To use a custom logo, select the 'Custom' option and specify an image file. Select 'Default' to restore the 'InetSoft' logo.
- **Favicon**: Specifies an image (generally 16x16 or 32x32) to be displayed in the browser tab for the User Portal. (Placement varies by browser.)
- **Font Family**: Specifies the font to be used for the content of the Portal page as well as any dialog boxes.
- **Viewsheet CSS**: Specifies a custom CSS file to use for Viewsheet component styles. The CSS file can specify styles for alignment, font, border, and color. See the *CSS Tab* section of *Advanced Dashboard Design* for more information.

To upload a custom Viewsheet CSS file, follow the steps below:

1. Select the 'Custom' option.
2. Click the 'Browse...' button. This opens the standard 'File Upload' dialog box.
3. In the 'File Upload' dialog box, locate your custom CSS file, and click 'Open'. The selected file is now displayed in the Viewsheet CSS text field.
4. Click 'Apply'. This uploads the specified CSS file to the server.

There can be only a single CSS file at any time.

The CSS file is saved as *{SREE Home}/portal/format.css*. You may need to restart the server for the new style settings to take effect.

Portal CSS Customization

Style Intelligence gives you the flexibility to implement your own custom User Portal. (See *Integrating the Web User Interface* in *Advanced Server Integration*.) However, in most cases it is easier and less time-consuming to customize the default User Portal using CSS styles.

To customize the Portal CSS, follow the steps below:

1. Clear the browser cache.

2. Navigate to the following location in the SREE Home directory (*sree/WEB-INF/classes*, by default, if you are using a file system data space):

   `sree/WEB-INF/classes/portal/css/`

   This directory structure contains *theme.css* and *tree-portal.css* files at the following levels:

   a. *sree/WEB-INF/classes/portal/css:* (*theme.css* and *tree-portal.css*)

   b. *sree/WEB-INF/classes/portal/css/modern*: (*tree-portal.css only*)

   c. *sree/WEB-INF/classes/portal/css/modern/granite*: (*theme.css* and *tree-portal.css*)

   Style definitions contained in files of the same name at different levels (for example, *portal/css/tree-portal.css, portal/css/modern/tree-portal.css, portal/css/modern/granite/tree-portal.css*) are merged together following CSS cascading rules.

3. Modify the desired *theme.css* and *tree-portal.css* files. The contents of the files are as follows:

   a. *theme.css*: Style definitions for the User Portal (excluding the Repository tree)

   b. *tree-portal.css*: Style definitions for all trees in the User Portal and Enterprise Manager

   For example, to display the “active” item in a Repository tree in bold with a gray background, modify the `JSTreeLabel_active` class in *portal/css//tree-portal.css* as follows:

```
.JSTreeLabel_active {
   font-size: 11px;
   font-weight: bold;
   background-color: #BBBBBB;
```

```
        white-space: nowrap;
    }
```

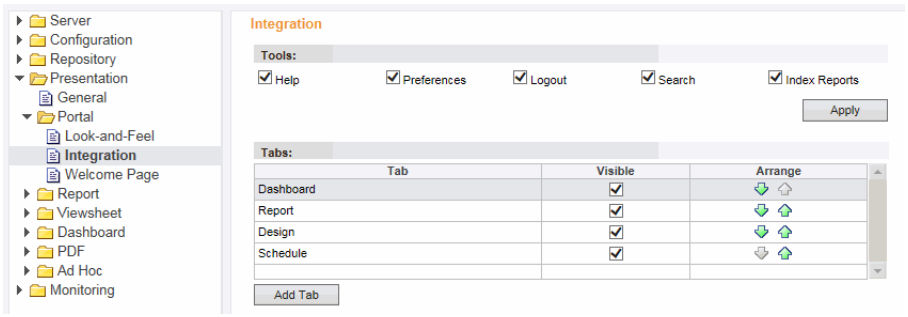
```
Note: You may need to restart the server after you modify CSS
files. Changes that you make to a CSS file may not be visible to
the user until the user's browser cache has been cleared.
```

4. Restart the server and clear the browser cache.

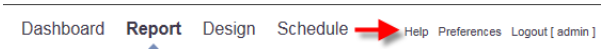
7.2.2 Integration

The tabs on the Portal page can be customized under this page. Administrators have the ability to add new tabs, hide one or more existing tabs, and specify the order of the tabs that are set to be visible. Additional tabs can be created by clicking on the 'Add Tab' button and specifying the tab name and the URI for its page.

Figure 38. Portal Integration Page



The 'Help', 'Preferences', 'Logout', and 'Search' options determine which of these features are available in the top-right corner of the User Portal.



The 'Logout' option is only in effect when security is enabled. The 'Search' option is only in effect when archiving is turned on, and enables complete searching within archived reports. The 'Index Reports' option enables searching of non-archived reports as well, but searches are limited to the report name, alias, description, keywords, and comments.

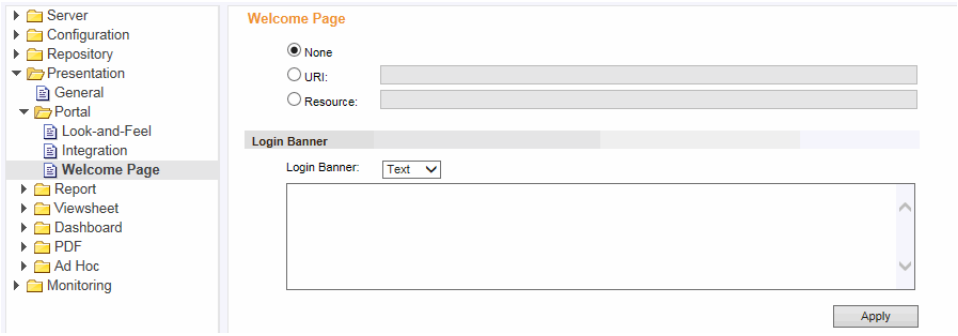
See Also

Archive, for more information about archiving.

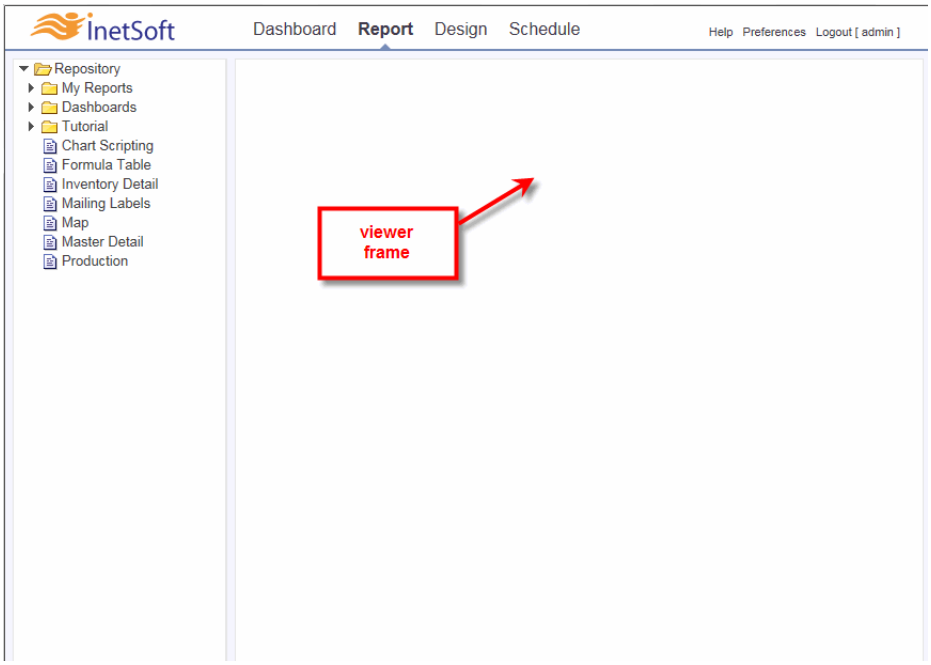
Advanced Server Integration, for more advanced approaches to web integration.

7.2.3 Welcome Page

The 'Welcome Page' screen allows you to set a home page to appear in the viewer frame of the Portal.



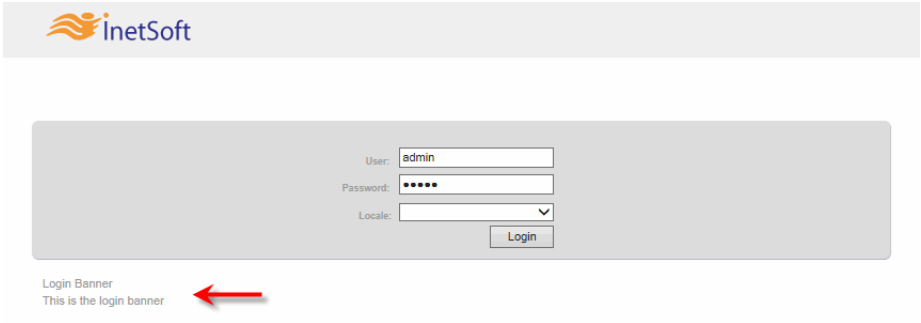
If you do not specify a welcome page, the viewer frame will remain blank.



You can specify a URL or a specific resource file such as an HTML or JSP file. HTML content can be localized by using the following tag:

```
$(text to be localized)#(inter)
```

The 'Login Banner' specifies text or HTML to be displayed below the password field of the Portal login page.



7.3 Report

The ‘Report’ pages allow you to configure options for report display and report interaction.

7.3.1 Viewer Options

These options allow you to tailor the appearance of the web-based report repository. Those options dealing with the report itself, such as the toolbar options, are applicable to both the Portal interface and the tree interface, whereas the other options only apply to the tree interface.

Table 3. Browser Options

OPTION	DESCRIPTION
Parameter Dialog HTML Template	Specify an HTML template that will be used for a parameter dialog box. See *Parameter Dialog HTML Template* below.
Parameter Dialog CSS File	Specify a CSS file used to generate the Parameter dialog box. See *Parameter Dialog CSS File* below.
Custom Replet Error Page	Specify a custom page that will display for a report error. See *Custom Replet Error Page* below.
Tool Dialog CSS File	Specify a CSS file used to generate the tool dialog boxes. See *Tool Dialog CSS File* below.
Repository Servlet Title	Specify a Portal page title. Localization is only applied if the default title is used.
Repository Header	The repository will display the header you define.
Repository Footer	The repository will display the footer you define.
Page Footer	A report will display a footer you define.
Page Background	Set the background color.
Report Session Timeout	Set the report viewing timeout interval for an inactive user (in seconds). Upon expiration, the user will see a message stating, “Session timeout. Please re-open the report.”
Concurrent User Session Timeout	For a server with session licence, set the timeout interval for an inactive user (in seconds).
Toolbar Style	Choose button toolbar options or text toolbar options.

Show Grouping Headers in Explorer	If selected, group headers in sections and/or tables will be included in the Report Explorer.
Enable Add to Queue	Presents users with an 'Add to Queue' button on the report loading screen.
Toolbar Visibility	Show or hide the report toolbar.
Toolbar Location	The toolbar can be positioned if css-p layout method is used. The possible values are: Top, Bottom, Left, Right, Top and Bottom.
Toolbar Background	Set the toolbar's background color.
Toolbar Font Color	Set the toolbar's foreground color.

Parameter Dialog HTML Template

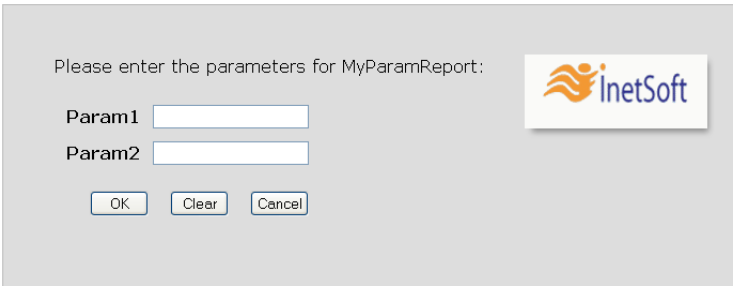
The 'Parameter Dialog HTML Template' field specifies an HTML template file (as a resource) to be used for the default parameter prompting screen. The template can contain regular HTML as well as three optional tags, `$(report.parameters)`, `$(report.title)`, and `$(report.description)`. These tags correspond, respectively, to the report parameter fields, report title, and report description. When the report prompts the user for parameters, it will load this HTML template and substitute the appropriate values for the tags.

A sample HTML template file is shown below:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//
EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en"
lang="en">
  <head>
  </head>
  <body>
    <img id='Logo' src='http://chart.inetsoft.com/images/
inetsoft.png'>
    <p>Please enter the parameters for $(report.title):</p>
    <div class='ParameterBlock'>
      $(report.parameters)
    </div>
  </body>
</html>
```

If you save this HTML template file with the name '*ParamTemplate.html*' in the SREE Home directory (*WEB-INF/classes*), you can enter the resource path in the 'Parameter Dialog HTML Template' field as follows: "`/ParamTemplate.html`".



Style can be added to the parameter screen by specifying an external CSS file in the 'Parameter Dialog CSS File' field. See *Parameter Dialog CSS File* for more details.

See Also

Parameter Sheets, in *Advanced Report Design*, for more extensive customization of parameter prompting.

Parameter Dialog CSS File

The 'Parameter Dialog CSS File' field specifies a CSS file (as a resource) to be used to style the parameter prompting screen. The CSS file can contain any ID or Class that is present in the parameter prompting screen; for example, `body{}` or `option{}`. This feature is most useful in conjunction with a *Parameter Dialog HTML Template*.

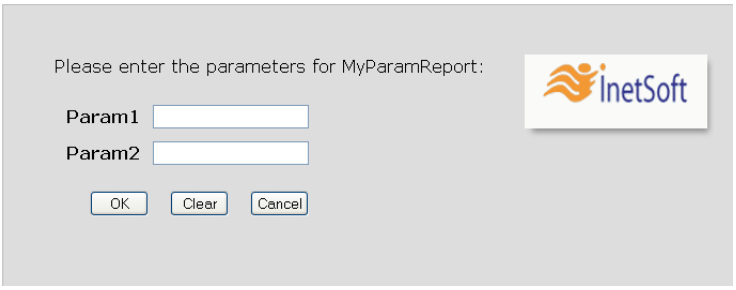
A sample CSS file which corresponds to the custom HTML template in the *Parameter Dialog HTML Template* section is shown below:

```
body {
  background-color:#DDD;
  font-family:Verdana;
  font-size:24;
  margin-left:3em;
  margin-top:3em;
}

img {
  float:right;
  margin-right:2em;
  padding:10px;
  border:2px solid;
  box-shadow: 5px 5px 5px #888;
}

div.ParameterBlock td {
  font-weight:bold;
}
```

If you save this CSS file with the name '*ParamStyle.html*' in the SREE Home directory (*WEB-INF/classes*), you can enter the resource path in the 'Parameter Dialog CSS File' field as follows: "`/ParamStyle.html`".



See Also

Parameter Sheets, in *Advanced Report Design*, for more extensive customization of parameter prompting.

Custom Replet Error Page

A replet error page is like any html page. Additionally, it can include two special tags, `$(replet.name)` and `$(replet.error)`, which are replaced inline with text containing the report name and the type of report error.

For example, the following is a minimal custom report error page:

```
<html>
<head>
  <title>Replet Error Page</title>
</head>
<body>
  $(replet.name) has error $(replet.error)
</body>
</html>
```

Tool Dialog CSS File

A CSS file can be used to style the dialog boxes generated by the Portal toolbar items. This CSS file is loaded as a resource, and the following classes and ids are supported:

DIALOG	CLASSES OR IDS
all	body{}, select{}, option{}, label{}, input.button{}, input.textbox{}
Find	#find_scope{}, #find_column{}, #find_operator{}, #find_logicOperator{}, #find_value{}, #find_moreButton{}, #find_fewerButton{}, #find_findButton{}, #find_clearButton{}, #find_cancelButton{}, #find_body{}
Export	#export_formatLabel{}, #export_formatList{}, #export_cancelButton{}, #export_exportButton{}, #export_pageLabel{}, #export_pageRadio{}, #export_pageScope{}, #export_body{}
Mail	#mail_toLabel{}, #mail_subjectLabel{}, #mail_formatLabel{}, #mail_format{}, #mail_to{}, #mail_subject{}, #mail_msg{}, #mail_okButton{}, #mail_cancelButton{}, #mail_body{}

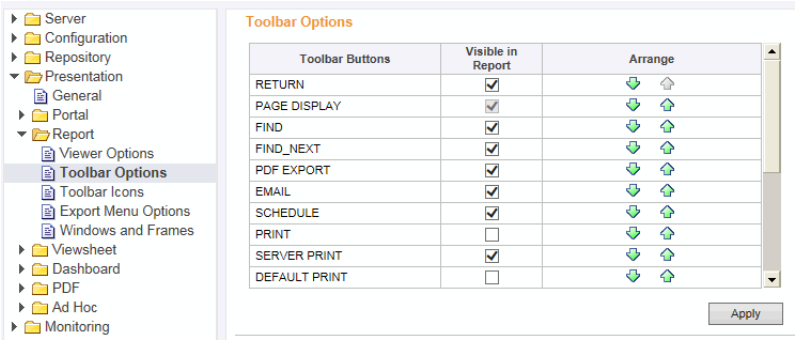
Print	#print_printerLabel{}, #print_printer{}, #print_selectPageLabel{}, #print_selectPageRadio{}, #print_pageScope{}, #print_okButton{}, #print_cancelButton{}, #print_body{}
Archive	#archive_title{}, #archive_secondTitle{}, #archive_label{}, #archive_text{}, #archive_button{}, #archive_comments{}, #archive_body{}

7.3.2 Report Toolbar Options

Use the 'Report' > 'Toolbar Options' page to hide or rearrange particular report toolbar buttons. To hide a toolbar button for reports viewed under the **Reports** tab, deselect the corresponding box in the 'Visible in Report' column.

To change the order of the toolbar buttons, click the up and down arrow keys. Click 'Apply' to save any changes.

Figure 39. Report Toolbar Options



Note that 'Default Print' button does not work with the built-in Flash plug-in of the Google Chrome browser. To use this button in Chrome, the user must disabled their browser's built-in Flash plug-in.

7.3.3 Report Toolbar Icons

To change the icons used for report toolbar buttons and the repository tree, select 'Presentation' > 'Report' > 'Toolbar Icons' under the **Server** tab, and specify an absolute path to the directory containing the custom graphics. Instead of using an absolute path, you can also use an environment variable such as '`sree.home`'. For example, you can specify *$(sree.home)/myIcons/*.

To prevent custom buttons from appearing on an individual report, set the string initialization parameter `use.custom.actions` to `false`.

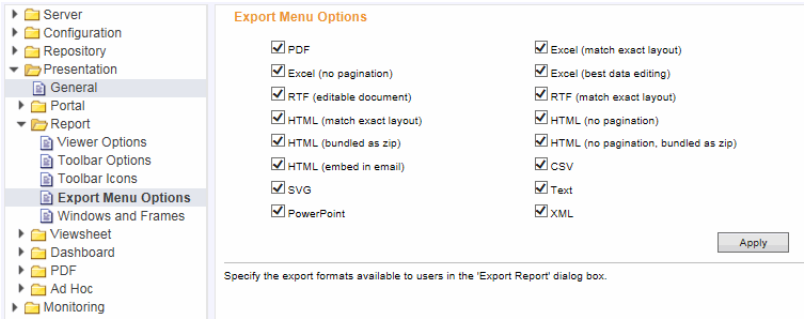
See Also

Viewsheet Toolbar Icons, to set custom icons for the dashboard toolbar.

7.3.4 Report Export Menu Options

The export menu on the viewer contains a list of all export file formats, including Excel, RTF, HTML, and others. For those three formats, there are two variations of output: one with report pagination, and another with no pagination.

It is possible to select a subset of the export formats to be included in the export menu. The export menu option is accessed through the 'Presentation' > 'Report' > 'Export Menu Options' node on the navigation tree.



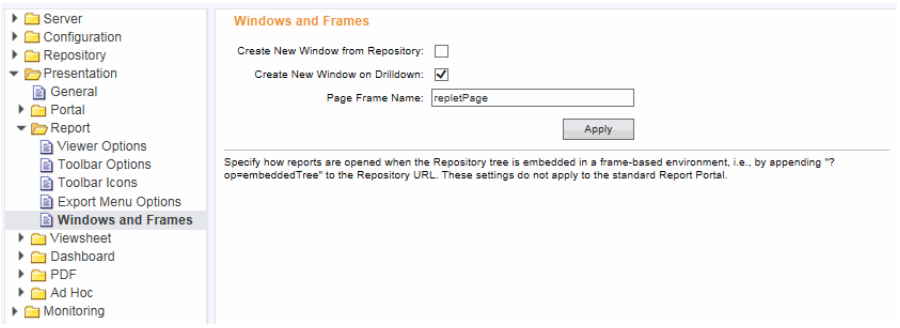
See Also

Viewsheet Export Menu Options, to control export options for a Viewsheet.
Options Tab, to override 'Export Menu Options' settings for a specific report.

7.3.5 Windows and Frames

Options can be set to control how and where to display reports.

Figure 40. Windows and Frames Options



The 'Page Frame Name' property allows the user to specify the frame in which the report pages show up, in cases where frames are being used. This option is most useful when a user wants to embed the Portal tree in their own customized interface. A user can follow the procedure listed below to embed the Portal tree in their own customized html page with frames.

1. Specify the target frame name in which to show the report pages, for example, ‘MyPage’. This is done by setting the ‘Page Frame Name’ property to “MyPage”.

2. Create an html page with contents similar to the following example.

```
<html>
<head>
   <title>Embedding Portal tree in my own html page</title>
</head>
<frameset name='mySet' cols='25%,*' frameborder='no'
border='0' framespacing='2'>
   <frame name='repository' scrolling='auto' src='http://
   {host}{:port}/{repositoryservlet}?op=embeddedTree'/>
   <frame name='MyPage' scrolling='auto' src=''/>
</frameset>
</html>
```

3. Deploy the customized html page to the web server and use it to access the repository tree.

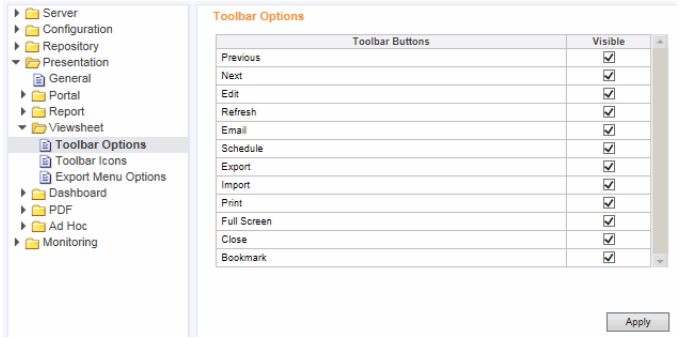
7.4 Viewsheet

The ‘Viewsheet’ pages allow you to configure options for Viewsheet display and interaction.

7.4.1 Viewsheet Toolbar Options

Use the ‘Viewsheet’ > ‘Toolbar Options’ page to hide particular Viewsheet toolbar buttons. To hide a button, deselect the corresponding box. Click ‘Apply’ to save changes.

Figure 41. Viewsheet Toolbar Options



The ‘Help’ option enables or disables the display of the ‘Help’ button in the Chart Editor.

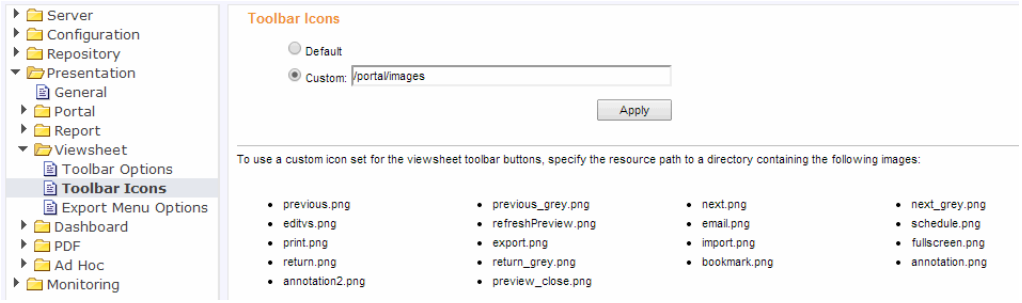
See Also

Report Toolbar Options, to modify the buttons on the report toolbar.

7.4.2 Viewsheet Toolbar Icons

Use the ‘Presentation’ > ‘Viewsheet’ > ‘Toolbar Icons’ page under the **Server** tab to specify an alternate set of icons for dashboard toolbars. In the ‘Custom’ field, enter an absolute path to the directory containing the custom graphics or a resource path to a directory on the classpath containing the images. For example, if your images are located in the *sree/WEB-INF/classes/portal/images* directory, specify the location as “`/portal/images`”. You can also use an environment variable such as `sree.home`, for example, `$(sree.home)/myIcons/`.

Figure 42. Viewsheet Toolbar Icons



The required file names are listed on the ‘Toolbar Icons’ page, and the recommended image size is a 21-pixel square. If a custom image is not found for a particular button, the default button image is used.

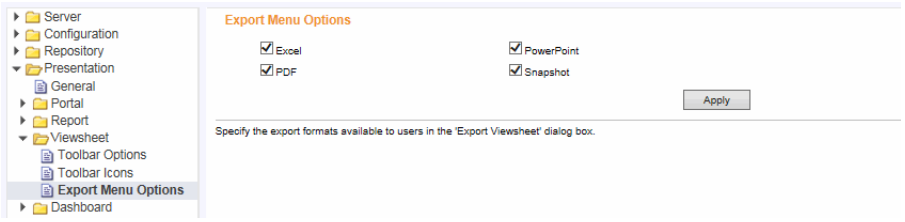
See Also

Report Toolbar Icons, to modify the buttons on the report toolbar.

7.4.3 Viewsheet Export Menu Options

Use the ‘Viewsheet’ > ‘Export Menu Options’ page to hide particular Viewsheet export options. To hide an export option, deselect the corresponding box. Click ‘Apply’ to save changes.

Figure 43. Viewsheet Export Menu Options



See Also

Report Export Menu Options, to control export options for reports.

7.5 Dashboards

Dashboards are used for monitoring key performance indicators, as well as for exploratory data analysis. Dashboards provide users with a variety of different views of their data using elements such as charts, tables, and gauges. They may additionally provide "input" control elements to allow users to interact directly with the data and tailor data views to their own needs.

In general, you will use the Enterprise Manager dashboard design tools to create "default" dashboards for end-users. Users can clone and adapt these dashboards to suit their own needs, as described in *Configuring Portal Preferences*. When a user-defined dashboard has the same name as an administrator-defined dashboard (for example, if the user clones a default dashboard), only the user-defined dashboard is shown.

See Also
Advanced Dashboard Design, for information on designing exploratory dashboards.
Advanced Report Design, for information on designing reports.
Deploying a Dashboard, for information on user-created dashboards.

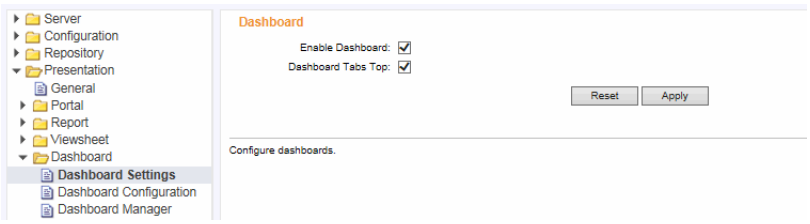
7.5.1 Enabling Dashboards

To enable dashboards for users, follow these steps:

1. Open the 'Dashboard Setting' page (in the **Server** tab, under the 'Presentation' > 'Dashboard').

2. On the 'Dashboard Setting' page, select 'Enable Dashboard'. This will enable the **Dashboard** tab in the User Portal.

3. (Optional) Select 'Dashboard Tabs Top' to place the individual dashboard tabs at the top of the Portal rather than at the bottom.



4. Click 'Apply' to enter the settings.

7.5.2 Creating a Dashboard

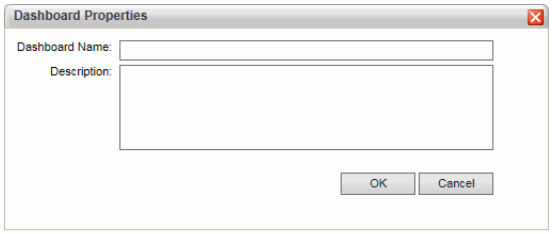
A dashboard consists of one or more Viewsheets. To create a new dashboard, follow these steps:

1. Open the ‘Dashboard Configuration’ page (under the **Server** tab in ‘Presentation’ > ‘Dashboard’).

2. Press the ‘New Dashboard’ button under the ‘Dashboards’ table. This opens the ‘Dashboard Properties’ dialog box.

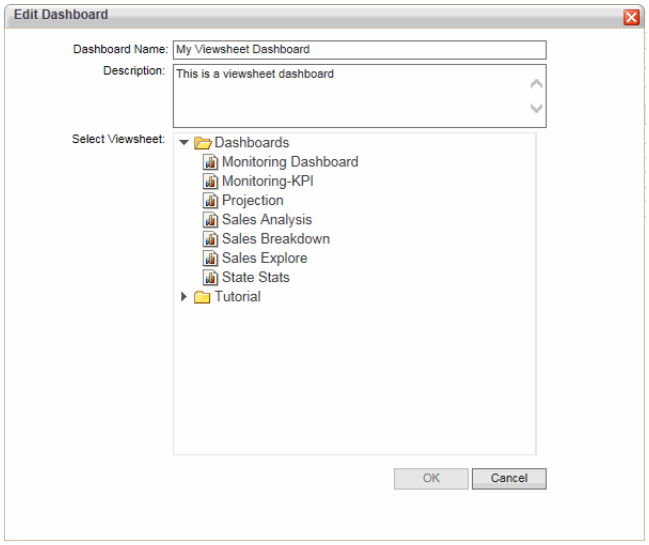
3. In the ‘Dashboard Name’ field, enter a name for the new dashboard. This name will be displayed on this dashboard’s tab in the User Portal.

4. (Optional) Enter a description for the dashboard in the ‘Description’ field. This description is visible when the dashboard is being edited by administrator or user.



5. Press ‘OK’ to open the ‘Edit Dashboard’ dialog box.

6. In the ‘Edit Dashboard’ dialog box, choose a Viewsheet from the ‘Select Viewsheet’ list. This Viewsheet will be used as the dashboard.



7. Press ‘OK’. This closes the ‘Edit Dashboard’ dialog box, and returns you to the ‘Dashboard Configuration’ page.

The new dashboard is now listed in the ‘Dashboards’ table, and will be visible to the users who have appropriate permissions.

See Also

Editing or Deleting a Dashboard, to edit an existing dashboard.
Building a Composite Dashboard, in *Advanced Dashboard Design*, for building composite dashboards.
Deploying a Dashboard, for information on user-created dashboards.
Managing Dashboards, to assign dashboard permissions to different users.

7.5.3 Enabling Server-Side Update for a Dashboard

To force a Viewsheet to refresh its data at a specified interval, simply select the ‘Server-Side Update’ option in the ‘Viewsheet Options’ dialog box. (See *Setting Viewsheet Options* in *Advanced Dashboard Design* for more details.)

You can further customize the update behavior by configuring the Viewsheet to refresh in response to a particular external event. For example, your database can issue a refresh command to a Viewsheet in response to a database event such as a data update.

To customize server-side update, follow the steps below:

1. Add the following line into the `sree.properties` file:

   ```
   assetMonitor.enabled=true
   ```

2. Start the Style Intelligence server, and open Enterprise Manager.

3. In Enterprise Manager, open the 'Server' page under the 'Log' node (on the **Server** tab). Verify that the following lines (or equivalent) appear in the log:

```
Start RMI registry at: 1099
Rebind RMI Call thread.
DataChange server bound in RMI registry.
```

   This indicates that the DataChange server (responsible for server-side updates) is correctly running.

4. If you do not see the above output in the log, this may indicate a port conflict. In this case, set the `dataChange.rmi.port` property in `sree.properties` to a different port number.

5. Open the desired Viewsheet in Visual Composer. Open the 'Viewsheet Options' dialog box, and select the 'Server-Side Update' option. (See *Setting Viewsheet Options* in *Advanced Dashboard Design*.) In the 'Refresh Interval' field, enter the period (in seconds) at which the server should check for the external trigger event.

6. Create a `DataChangeClient` class that can be executed by an external program (for example, by the database) to trigger a particular Viewsheet to update. An example of such a class is shown below:

```
import java.rmi.*;
import inetsoft.report.composition.AssetMonitor;
import inetsoft.uql.asset.AssetRepository;

public class DataChangeClient {
  public DataChangeClient(String usr, String pwd, int
scope, String vs, String rmi) {
    this.usr = usr;
    this.pwd = pwd;
    this.scope = scope;
    this.vs = vs;
    this.rmi = rmi;
  }

  private boolean dataChange() {
    AssetMonitor engine = null;

    try {
      engine = (AssetMonitor) Naming.lookup(rmi);
    }
    catch(Exception e) {
      System.err.println("Can't find the rmi server at: " +
rmi);
      e.printStackTrace();
      return false;
    }

    String ticket = null;
```

```
    try {
      ticket = engine.login(usr, pwd);
    }
    catch(RemoteException e) {
      System.err.println("Login failed with " + usr + "/" +
pwd);
      e.printStackTrace();
      return false;
    }

    try {
      engine.dataChanged(vs, scope, usr, ticket);
    }
    catch(RemoteException e) {
      System.err.println("fire data change failed");
      e.printStackTrace();
      return false;
    }

    try {
      engine.logout(ticket);
    }
    catch(RemoteException e) {
    }
    return true;
  }

  public static void main(String[] args) {
    if(args.length != 5) {
      System.err.println("Usage: java DataChangeClient usr
pwd scope viewsheetPath rmi");
      System.err.println("@param user: the usr for login");
      System.err.println("@param pwd: the password of the
user");
      System.err.println("@param scope: 1 - global scope,
other - user scope");
      System.err.println("@param viewsheetPath: the path of
the viewsheet");
      System.err.println("@param rmi: the rmi data change
server url");
      System.exit(0);
    }

    int scope = AssetRepository.USER_SCOPE;
    if("1".equals(args[2])) {
      scope = AssetRepository.GLOBAL_SCOPE;
    }

    new DataChangeClient(args[0], args[1], scope, args[3],
args[4]).dataChange();
  }

  private String usr, pwd, vs, rmi;
  private int scope;
}
```

7. Compile the `DataChangeClient` class. (The classpath should include the Style Intelligence server files, *bisuite.jar* and *etools.jar*.)

8. Execute the `DataChangeClient` class to update the desired Viewsheet. For example:

```
java DataChangeClient admin admin 1 ViewsheetName "rmi://
localhost:1099/DataChangeServer"
```

This will update the specified Viewsheet after the interval specified by the 'Refresh delay' field in the 'Viewsheet Options' dialog box.

Example: Custom Server-Side Update

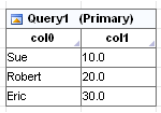
In the following example, you will create a Viewsheet that receives its data from an embedded Table in a Data Worksheet. You will then configure the Viewsheet to automatically update when the `DataChangeClient` is triggered.

Before beginning the example, compile the `DataChangeClient` class as described previously. Then follow the steps below:

1. Add the following line into the `sree.properties` file:

   ```
   assetMonitor.enabled=true
   ```

2. Create a new Data Worksheet, and add the following Embedded Table. (See *Creating an Embedded Table* in *Advanced Data Mashup* for detailed instructions.)

Query1 (Primary)	
col0	col1
Sue	10.0
Robert	20.0
Eric	30.0

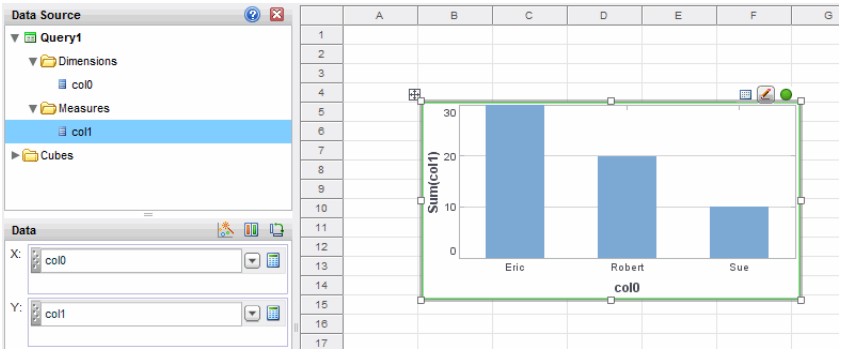
3. Right-click the 'col1' header, and select 'Column Type' from the context menu. This opens the 'Column Type' dialog box.

4. Select 'Double' from the 'Type' menu, and click 'OK'.

5. Save the Worksheet with the name 'EmbeddedData'. (See *Saving a Data Worksheet* in *Advanced Data Mashup*.)

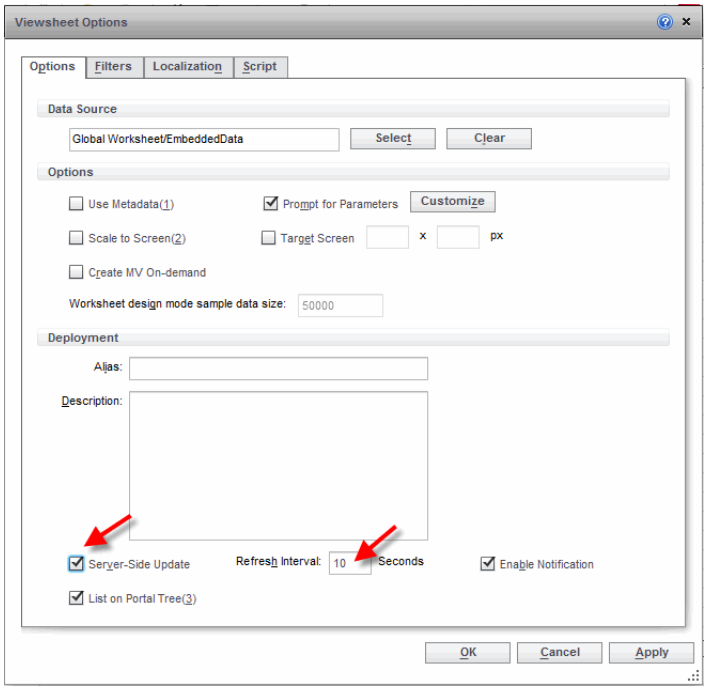
6. Create a new Viewsheet based on the 'EmbeddedData' Worksheet. (See *Creating a New Viewsheet* in *Advanced Dashboard Design*.)

7. Add a Chart to the Viewsheet to display the data from the Worksheet as follows. (See *Creating a Chart* in *Advanced Dashboard Design*.)



8. Click the ‘Options’ button in the Viewsheet toolbar. This opens the ‘Viewsheet Options’ dialog box.

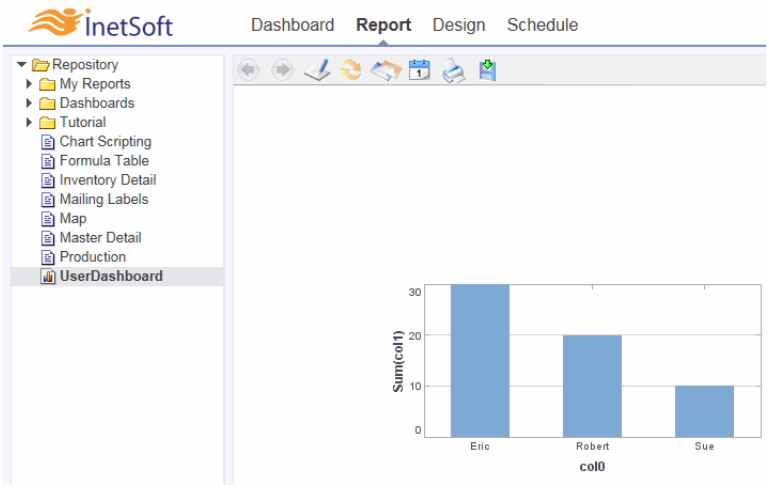
9. In the ‘Deployment’ panel at the bottom, enable the ‘Server-Side Update’ option, and set the ‘Refresh Interval’ to 10 seconds.



10. Click ‘OK’ to close the dialog box.

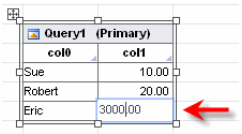
11. Save the Viewsheet with the name ‘UserDashboard’. (See *Saving a Viewsheet* in *Advanced Dashboard Design*.)

12. Launch the User Portal, and open the ‘UserDashboard’ Viewsheet in the Portal.



13. Return to the 'EmbeddedData' Worksheet in Visual Composer.

14. Make some changes to the data in the Embedded table, and click the 'Save' button.

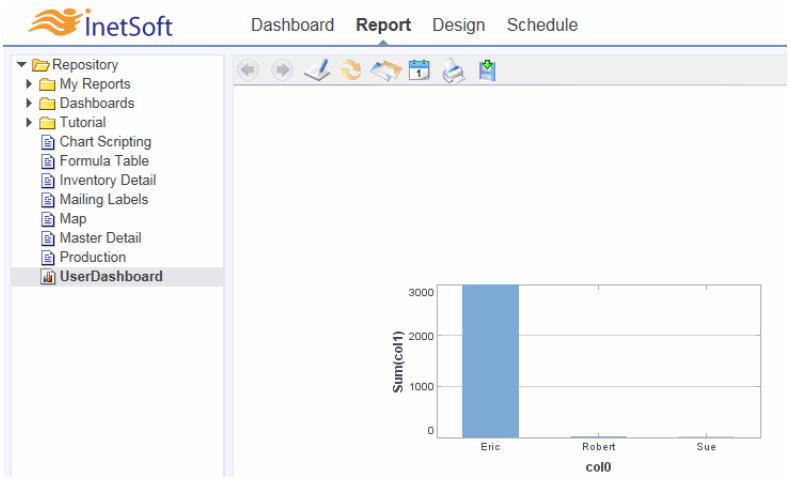


15. Return to the User Portal. (Do not reload the Viewsheet.) Note that the Viewsheet remains the same, even though the underlying data has changed.

16. Trigger an automatic Viewsheet update by running the `DataChangeClient` class from the command line:

```
java DataChangeClient admin admin 1 UserDashboard "rmi://
localhost:1099/DataChangeServer"
```

17. Return to the User Portal. (Do not reload the Viewsheet.) Observe that the Viewsheet updates automatically after the specified delay.



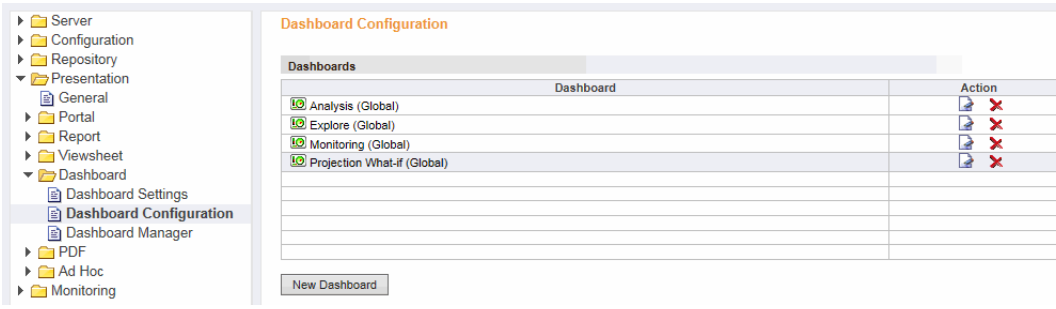
7.5.4 Editing or Deleting a Dashboard

To edit an existing dashboard, follow the steps below:

1. Open the 'Dashboard Configuration' page (in the **Server** tab, under 'Presentation' > 'Dashboard').

2. Locate the dashboard you wish to edit in the 'Dashboards' table. (This table lists all of the dashboards defined by the administrator. User-defined dashboards are not listed in this table.)

3. Click the 'Edit' button, located in the right-most column, for the dashboard you wish to edit.



This opens the 'Edit Dashboard' dialog box. Follow the instructions in *Creating a Dashboard* to make the desired edits.

To delete a dashboard entirely, click the 'Delete' button in the right-most column.

7.5.5 Managing Dashboards

Before a user can access the dashboards that you have created in Enterprise Manager, you need to assign the user appropriate permissions based on the

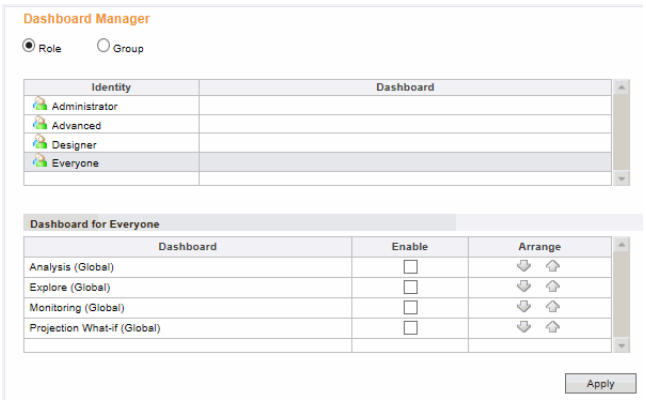
user's role or group. In general, you should first configure a security provider (see *Specifying a Security Provider*), otherwise, the only role available will be "anonymous".

To configure permissions for a dashboard, follow the steps below:

1. Open the 'Dashboard Manager' page (in the **Server** tab, under 'Presentation' > 'Dashboard').

2. Select 'Role' or 'Group', depending on the scope of permission you want to assign. (See *Users, Roles, and Groups* for more information.)

3. Click on an identity in the top table. This exposes the 'Dashboard for' table. For example, click the 'Everyone' role to expose the 'Dashboards for Everyone' table.



4. Check the box in the 'Enable' column next to the dashboards you wish to activate for the selected role.

5. Click the arrows in the 'Arrange' columns to position the User Portal dashboard tabs in the desired display order. (This ordering only affects newly added dashboards. After a user has already accessed a dashboard, the ordering for that dashboard is controlled by the user through the Portal 'Preferences' dialog box. See *Configuring Portal Preferences*.)

6. Click 'Apply' to enter the settings.

The table lists the dashboards that are currently accessible to a given role.



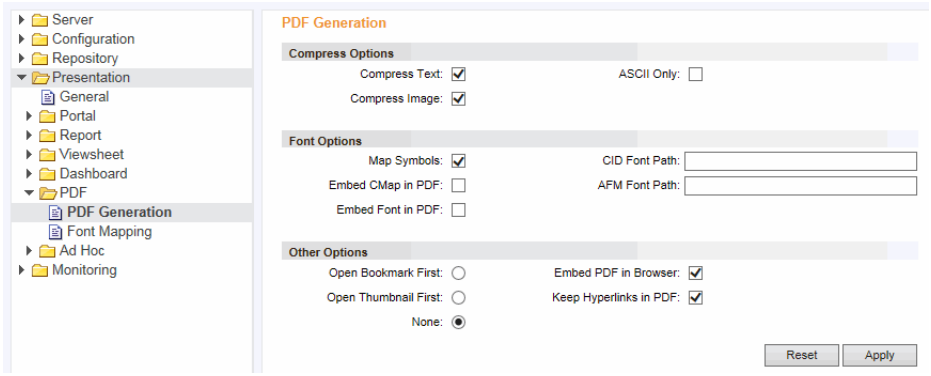
Dashboards that you enable for a user will be visible to the user the next time they access the **Dashboard** tab in the Portal. The enabled dashboards will also be appended to the 'Dashboard' table of the user's Portal 'Preferences' dialog box. Dashboards that you disable for a user will no longer be listed in this table. From the Portal 'Preferences' dialog box, users can choose to disable or reorder dashboards to suit their needs. See the *Deploying a Dashboard* for more information.

7.6 PDF Generation

PDF is used as the standard format for saving and presenting electronic report data. It is used to deliver reports via email and to view them with the browser. InetSoft software uses enhanced PDF generation that allows other fonts to be embedded in the PDF file.

7.6.1 Setting PDF Properties

You can set PDF properties through the 'PDF Generation' page of the Enterprise Manager. Select 'Presentation' > 'PDF' on the navigation tree under the **Server** tab to access this page.



Since PDF generation is built into the report engine, there is normally no need for replets to access the PDF generator API directly. The following sections describe the properties available through the 'PDF Generation' page. See Appendix B.9, *PDF and Font Properties*, for more information on PDF properties.

Select the 'Font Mapping' node to change the default base-14 font mapping.

- **Embed PDF**: If 'Embed PDF in Browser' is enabled, when the user exports a report or Viewsheet to PDF format from within the User Portal, the PDF file will be immediately displayed by the browser (if Adobe Reader is installed), and the 'Open/Save' download dialog box will not be displayed.

- **Open Bookmark**: If ‘Open Bookmark’ is set to true, then all the generated PDFs will have bookmarks displayed immediately upon opening them in Adobe Acrobat.
- **Open Thumbnail**: If ‘Open Thumbnail’ is set to true, then all the generated PDFs will have thumbnails displayed immediately upon opening them in Adobe Acrobat.
- **Keep Hyperlinks in PDF**: Setting PDF option ‘Keep Hyperlinks in PDF’ to ‘false’ will turn off hyperlink generation and speed up the PDF generation process. Otherwise, the hyperlinks will be created in the resulting PDF file.

‘Open Bookmark’ and ‘Open Thumbnail’ are mutually exclusive properties. When both are set to true, ‘Open Bookmark’ takes priority and only bookmarks are displayed when opening the PDF file in Adobe Acrobat.

7.6.2 TrueType Fonts

The TrueType font directories need to be specified in the `font.truetype.path` property. The property is a directory path and can contain multiple directories separated by a semicolon separator. Only TrueType fonts on this path are used in PDF generation. In Windows NT, TrueType fonts are stored in the *c:\winnt\fonts* directory. Under the UNIX platform, information about the available TrueType fonts is obtained from the `fonts.properties` file which is used by the JRE. As a result, if the custom installed TrueType fonts are to be used for PDF generation, then those fonts only need to be added to the `font.properties` file.

7.6.3 Type 1 Fonts

Type 1 fonts can also be embedded in PDF files. Type 1 font information is retrieved from AFM files. AFM is the standard font format used by Adobe to store font data, and AFM Files can be downloaded from the Adobe Web site. The PDF generator uses `font.afm.path` to search for AFM files for a Type 1 font. Applications using Type 1 fonts need to package the AFM files with the application, and set the `font.afm.path` to point to the AFM directory.

7.6.4 CJK Fonts

The PDF generator is capable of handling CJK (Chinese, Japanese, Korean) characters in generated PDF files. Currently supported CJK fonts include the following:

– AdobeMingStd-Light
– AdobeMingStd-Light-Acro
– AdobeMyungjoStd-Medium
– AdobeMyungjoStd-Medium-Acro
– AdobeSongStd-Light
– AdobeSongStd-Light-Acro
– HYGoThic-Medium-Acro
– HYSMyeongJo-Medium-Acro
– HYSMyeongJoStd-Medium-Acro
– HeiseiKakuGo-W5-Acro
– HeiseiMin-W3-Acro

– KozGoPro-Medium
– KozGoPro-Medium-Acro
– KozMinPro-Regular-Acro
– KozMinProVI-Regular
– MHei-Medium-Acro
– MSung-Light-Acro
– MSungStd-Light-Acro
– STSong-Light-Acro
– STSongStd-Light-Acro

Supporting CJK Characters

To support CJK characters:

- The location (folder path) of the CJK font files must be specified under the heading *TrueType/CID Font Path*.
- The font mapping from the CJK-TrueType font to a CJK-CID font needs to be defined in the font mapping table. It is recommended that you select 'true' for embedding the font and the cmap in the PDF file.

Note: CJK fonts often use separate font files for the plain and bold versions of the font. It is therefore possible to map a CJK font to a different file depending on its weight. For example, the following setting maps all plain MS Mincho instances to a normal-weight version of the KozMin font-family, and maps all bold MS Mincho instances to a bold-weight version of the same font-family:

```
pdf.font.mapping=MS Mincho-plain\:KozMinProVI-Regular;
MS Mincho-bold\:KozMinStd-Bold;
```

If the font weight is omitted, the default is '`-plain`'.

Supporting Unlisted CJK Fonts

To support CJK fonts not listed above, do the following:

- Place the font files in the folder specified under *TrueType/CID Font Path*.
- Extract the *cjkmap.properties* file from your build JAR file (*sree.jar* or *bisuite.jar*).
- Place the *cjkmap.properties* file within your classpath under */inetsoft/reports/pdf*, e.g., '*WEB-INF/classes/inetsoft/reports/pdf/cjkmap.properties*'.
- Make an entry for the CJK font as {font name}={encoding}, e.g., *DFKMincho\ Std\ W5=kor*. (Escape the white spaces using a backslash).

Note: In many cases the actual font name may not be the same as the font file name. To make sure you enter the right name in the properties file. Use

the utility *TTFontInfo*. The correct name will appear under the *fontnames[3]* heading.

```
java -cp sree.jar inetsoft.report.pdf.TTFontInfo
{FontFileName With Absolute Path}

e.g. java -cp sree.jar inetsoft.report.pdf.TTFontInfo c:/
CID/DFKMinchoStd-W5.otf

Sample Output:
Getting font names:[c:/CID/DFKMinchoStd-W5.otf]
fontnames[0]=DFKMincho Std W5
fontnames[1]=DFKMincho Std W5
fontnames[2]=DFKMinchoStd-W5
fontnames[3]=DFKMincho Std W5
```

Notice how `fontnames[3]` yields the right font name to enter into the *cjkmap.properties* file.

7.7 Adding PDF Security

To add security encryption to a PDF file exported by Style Intelligence, use the following script in the onLoad handler of the report.

```
docInfo.pdf.password.owner = '{owner_password}';
// owner_password provides owner access to document

docInfo.pdf.password.user = '{user_password}';
// user_password provides user access to document

docInfo.pdf.permission.print = true/false;
// provides the ability to print

docInfo.pdf.permission.copy = true/false;
// provides the ability to copy from the document
```

To use encryption, you must have an implementation of the Java Cryptography Extension on your classpath. Sun provides a default implementation with the JRE. To use this, add the {JAVA_HOME}/jre/lib/security folder to your classpath. See *PDF Security Provider Subsets*, in *Report Scripting*, for more information.

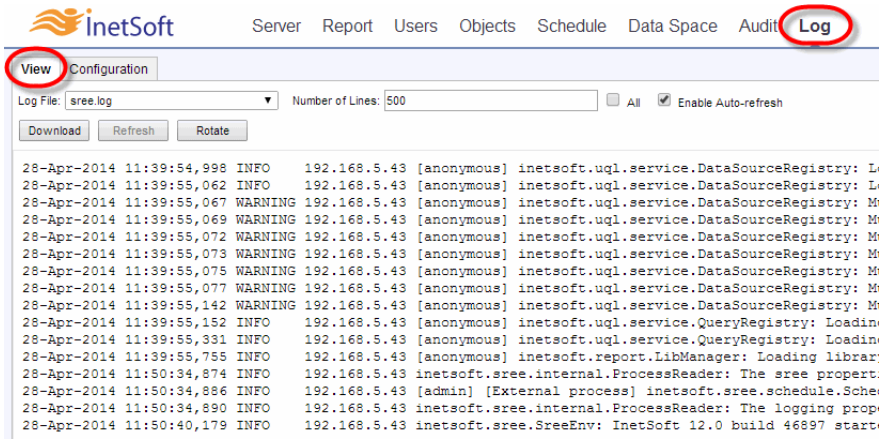
8 Logging

A record of server operations is stored in the *sree.log* file in the server repository directory (*WEB-INF/classes*, by default). A record of Scheduler operations is stored in the *schedule.log* file in the same default location. See Appendix B.6, *Scheduling Service Properties*, for details on how to specify a different location for this file.

8.1 Viewing and Downloading Log Files

To view the log files in Enterprise Manager, follow the steps below:

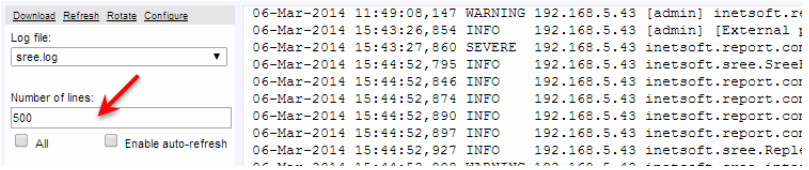
1. Select the main **Log** tab in Enterprise Manager, and select the **View** tab.



If the log size has exceeded the 'Maximum Log File Size' set on the **Configuration** tab, you will see multiple log files listed.

2. From the 'Log file' menu, select '`sree.log`' for the server log or '`schedule.log`' for the Scheduler log. This will display the selected log.

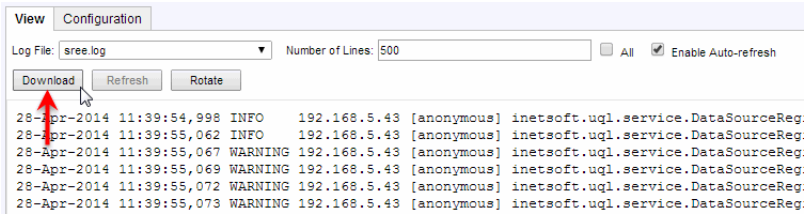
3. Select the 'All' option to display the entire contents of the log file (maximum 3000 lines by default), or enter the 'Number of Lines' from the end of the file to display.



`Note: These settings are not retained following a server restart.`

4. Select 'Enable Auto-refresh' to allow the right panel to update automatically as new lines are written to the log.

5. To download a zip archive containing all log files, press the ‘Download’ button at the top of the page.



6. To refresh the display with current log records, press the ‘Refresh’ button (only available when ‘Enable Auto-refresh’ is disabled).

7. To start a new log file, press the ‘Rotate’ button. This saves the current log file under an automatically-generated name such as *sree.log.1* or *schedule.log.2*, and begins adding log records to a new log file.

To configure the log settings, select the **Configure** tab. See *Configuring Logging* for more information.

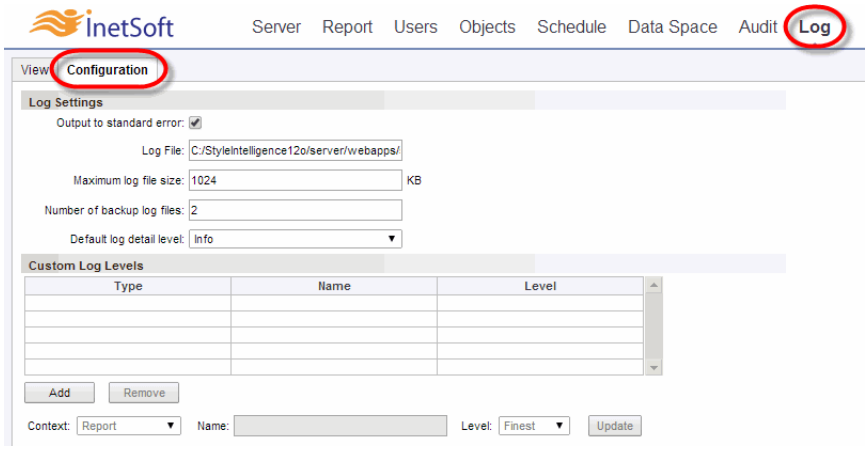
See Also

Configuring Logging, for information about configuring logging behavior.

8.2 Configuring Logging

Enterprise Manager provides the ability to log messages generated during server and Scheduler execution. Logged messages are appended to a text file (default: *sree.log* for the server, and *schedule.log* for the Scheduler) and optionally sent to “`stderr`”. For information on viewing log files, see *Viewing and Downloading Log Files*.

To configure logging settings, select the **Configuration** tab under the main **Log** tab.



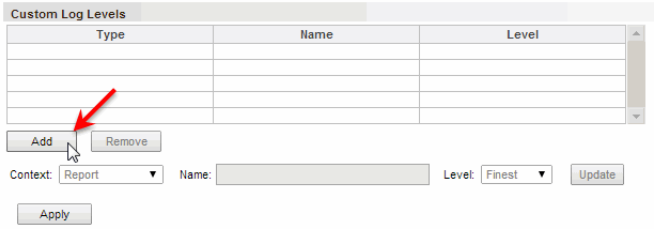
The following properties are available under the ‘Log Settings’ panel.

SETTING	DESCRIPTION
Output To Standard Error	When enabled, sends log messages to the standard error output stream.
Log File	The location and base name of the log file.
Maximum Log File Size	The maximum size (KB) to which the log file should be allowed to grow before it is rotated. To rotate the log file manually, see *Viewing and Downloading Log Files*.
Number of Backup Log Files	Specifies the number of rotated log files to maintain (e.g., *sree.log.1*, *sree.log.2*, etc.). Backup log files are created when the size of the log exceeds the ‘Maximum Log File Size’ property or the log is rotated manually.
Default Log Detail Level	There are five available detail levels: ‘Finest’ provides internal execution-tracing information for debugging purposes only. ‘Fine’ reports detailed trace information for use when diagnosing a particular problem, generally in collaboration with InetSoft Technical Support. ‘Info’ reports information about the state of the server, such as the security provider in use and the location of the *sree.properties* file. ‘Warning’ reports errors that do not prevent normal operation, but indicate a possible problem that should be investigated. ‘Severe’ reports errors that prevent normal operation. It is recommended that you set the detail level to ‘Warning’ or ‘Info’ for regular use.

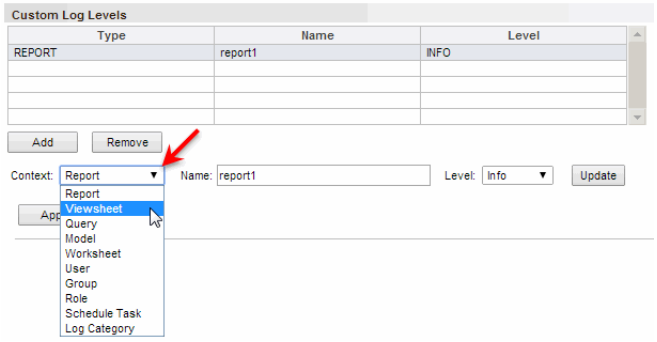
You can assign custom log levels to particular assets. For example, you can set a ‘Warning’ log level for the ‘Default Log Detail Level’ property, but still set a ‘Fine’ log level for a particular report.

To set a custom log level, follow the steps below:

1. Press the ‘Add’ button below the table. This adds a new log level.

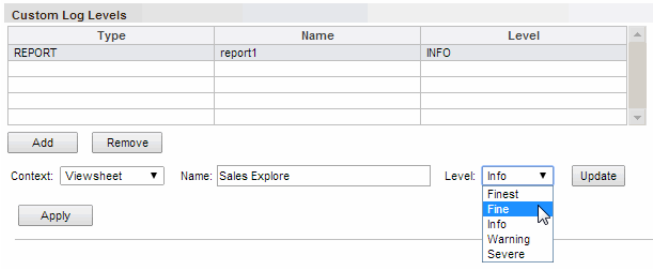


2. In the ‘Context’ menu, select the type of asset (e.g., report, Viewsheet, query, etc.) to which you want to assign a distinct log level.



3. In the ‘Name’ field, enter the name of the asset for which you want to set a custom log level.

4. Select the desired logging level for this particular asset. (See above for level information.)



5. Press the ‘Update’ button, and then press the ‘Apply’ button.

To remove a custom log level, select the custom log level in the table and press the ‘Remove’ button below the table.

8.2.1 Advanced Log Configuration

Logging in Style Intelligence is implemented by the standard Java Logging API of the `java.util.logging` package. Log messages can be customized and formatted without implementing a custom logging mechanism. You can selectively set different log levels for different packages, and include useful context based information within logs.

```
Note: If the report server cannot locate the ‘logging.properties’
file, basic logging properties will be used (from sree.properties).
```

To customize your logs, include a ‘`logging.properties`’ file for the server log and a ‘`schedule.logging.properties`’ file for the scheduler log within your SREE Home directory.

A sample ‘`logging.properties`’/‘`schedule.logging.properties`’ file is show below:

```
.level=INFO
handlers=java.util.logging.ConsoleHandler,
inetsoft.util.log.FileHandler
inetsoft.util.log.FileHandler.fileName=${sree.home}/
sree.log
inetsoft.util.log.FileHandler.limit=1048576
inetsoft.util.log.FileHandler.count=2
inetsoft.util.log.FileHandler.formatter=inetsoft.util.log.
PatternFormatter
inetsoft.util.log.FileHandler.formatter.pattern=%l %dd-
%db-%dY %dH:%dM:%dS %u %c %M %m
inetsoft.sree.web.level=FINE
inetsoft.uql.jdbc.level=FINE
```

- The first property sets the default log level (`FINEST`, `FINE`, `INFO`, `WARNING`, or `SEVERE`).
- The second property sets the different log handlers. You can implement your own custom handler by implementing the `java.util.logging.Logger` interface.
- The '`fileName`', '`limit`', '`count`' and properties specify the log file name, size limit, and the number of files to archive.
- The `formatter.pattern` specifies the format of the log message using the following characters:

```
%l              Log level
%dd,%db,%dY     Log Date (Day, Month, Year)
%dH:%dM:%dS     Log Time (Hour, Minute, Second)
%u              User Name
%c              Class Name
%M              Class Method
%m              Log Message
```

- The '`level`' property specifies the log detail lever for a specific package. This lets you assign finer logging for certain modules, without cluttering the log with messages from other modules.

8.2.2 Writing Log Messages in Custom Code

Use the `java.util.logging.Logger` class to write log messages in your custom implementations, e.g., Custom Security Provider.

Use the `getLogger(String name)` method to get an instance of the logger, for example,

```
private static final Logger logger =
Logger.getLog(MyClass.class.getName());
```

To check if a certain level is loggable, use the `inetsoft.util.LogUtil` class, for example,

```
if(LogUtil.isLoggable(logger, Level.TRACE)) {
   logger.log(Level.WARNING, "Task failed", excp);
}
else {
   logger.log(Level.WARNING,"Task failed");
```

```
}
```

8.2.3 Implementing a Custom Log Handler

```
Note: As of version 10.2, Style Intelligence uses standard Java
logging. The ReportLog.Delegate interface has been deprecated, and
should no longer be used.
```

To implement a custom log handler, use a class that implements the `java.util.logging.Logger` interface. Register the class by adding it to the '`logging.properties`' file under the '`handlers`' entry. See *Writing Log Messages in Custom Code* for more details.

APPENDIX A: Troubleshooting

None of my settings have taken effect.

If your configuration files are in a folder other than the `WEB-INF/classes` folder of the web application, a `sree.home` initialization parameter must be set for both the administrator and the repository servlets.

I cannot see my reports after I deploy them.

This could be a problem with the location of the repository directory or the CLASSPATH in the application server.

This problem could also occur if you have security set up for your environment. If you have set up security and have created users and roles, then the Report Server will require a username and password ticket to generate the replets. The users must have permission to view the replets.

I can see my reports in the repository tree but when I try to view them I get a "replet not found" error.

All report files (.srt) and class files are loaded from the classpath of the application server. Make sure that these files are located in a folder which is in the CLASSPATH (or any of its sub-folders). If the replets are deployed from a packaged JAR file, make sure that the `replet.auto.reload` property is set to false.

I deployed a template but I cannot see it.

Templates are loaded as resources. When a template is registered it must be prefixed with a '`/`', for example '`/template1.xml`'. Also, be sure to select 'Template' as the type on the registration page.

I can see the body of my report, but the fields are not populated with data.

Check the location of the data source and query registry files in the Enterprise Manager. Make sure that valid *datasource.xml* and *query.xml* registry files are referenced.

My Charts seem to have jagged edges, especially pie charts.

Try changing the image type to PNG rather than JPEG using the 'Presentation' > 'General' node in the Enterprise Manager.

The Repository does not bind to the RMIREGISTRY.

Make sure the classpath is set before running the `rmiregistry`. Alternatively, make sure that the port is not in use, and it matches the parameter value.

The RMI node will not start.

Make sure that the rmiregistry and the Activator are running on the node.

The repository does not bind to the name service.

Make sure that the same port is being used.

Is it possible to run a task continuously (i.e., create a task that executes every few minutes)?

It is possible to do this by writing a user-defined `ScheduleAction` which extends `DefaultUserAction`. A scheduled task consists of one or more actions and one or more conditions. The conditions are checked to see when and whether a task will be started. When all conditions are met, the task actions are executed in order. If a task is a repeated task, it will be rescheduled to run at the next cycle. Multiple time conditions typically should not be added for the same scheduled tasks (this can result in retry time conflicts); instead, another task should be created.

This software will not run in WebSphere.

WebSphere has an XML parser (*WebSphere\AppServer\lib\xml4j.jar*) which uses DOM level 1 interfaces. Even if you put the Xerces 1.1.2 (or later) *xerces.jar* in your classpath, the wrong interfaces are found by any Java code running in WebSphere, because WebSphere puts the user's classpath at the end. Therefore, you must edit the *WebSphere\AppServer\bin\admin.config* file and put *xerces.jar* at the *beginning* of the `com.ibm.ejs.sm.adminserver.classpath` variable. This software uses the newer `xerces.jar` parser – therefore, it is necessary to edit this property so the servlet can be deployed without error.

CVS Archive will not work with Sun ONE (iPlanet) application server.

The Sun ONE application server is unable to use the `cvs` executable on a remote drive. Move the executable file to a local drive.

APPENDIX B: Configuration Properties

This Appendix lists all properties used by this software's components. Properties are used in the report engine to control the operation of the report formatting and printing. You have encountered the use of properties in previous chapters, such as the `StyleReport.useCustomDriver` property for controlling the printer driver selection.

Properties are set using the `SreeEnv` class, which has static methods for setting and retrieving the property values. The `SreeEnv` class uses the system properties as the base property setting. Any properties set in the system properties, through command line or Java API, are accessible from the `SreeEnv` class.

Most of the properties have a corresponding component in the Enterprise Manager. For normal use, we recommend using the Enterprise Manager to change the property values. Misconfiguration of property values could cause the components to malfunction. The property files can be shipped as part of an application and customized during the installation. Alternatively, an installation script can be used to modify the property files to suit the server environment in batch mode. The advantage of using a script is that no user intervention is needed during the installation, and the installation of the application can be completely automated. Because all configuration files are plain-text based, any scripting language can be used to develop the installation scripts.

B.1 Server Engine Properties

Server engine properties include specifications relating to configuration file locations and names. See *Specifying Data Source Information* for related properties in Enterprise Manager.

PROPERTY	DESCRIPTION
StyleReport.ditherImage	True to force all images to be dithered before printing. Default is false.
StyleReport.dither	True to dither image when printing to black/ white printer. Default is false.
StyleReport.useCustomDriver	True to use custom driver. Default is false.
StyleReport.locale.resource	Locale mapping file resource name.
StyleReport.locale.properties	Locale mapping property file name.
StyleReport.locale.encode	Load the locale property file used by Catalog, so other language text can be properly loaded.
stylereport.library.file	Location of `.srl` (Style Report Library) file.

stylereport.library.path	List of `.srl` files specified using the file's full path. Use this when using multiple `.srl` files.
datasource.registry.file	Location of datasource.xml file.
datasource.registry.path	List of `.xml` files that are used to define all of the available data sources.
datasource.selectedSource	Specifies the datasource for which to refresh metadata. See *Specifying Data Source Information* to set this property in Enterprise Manager.
query.registry.file	Location of `query.xml` file.
query.registry.path	List of `.xml` files that are used to define all of the available queries.
query.cache.data	Set to `true` (default) to cache query result sets for reuse across reports, Worksheets, and Viewsheets.
query.cache.timeout	With `query.cache.data` enabled, specifies the amount of time (ms) that result sets for report, Worksheet, and Viewsheet queries persist in the cache. Default: 30 seconds for reports and 10 minutes for Worksheets/ Viewsheets.
query.cache.limit	With `query.cache.data` enabled, specifies the number of query result sets to be retained in the cache. Default: 100 datasets (for both reports and Worksheets/Viewsheets).
javascript.java.packages	List of JavaScript packages accessible to the scripting environment.
composer.comm.error	Set to `true` (default) to warn user about server connection errors when using Viewsheets. Set to false to suppress these warnings.

B.2 DHTML Generation Properties

DHTML generation properties include meta information settings for the HTML pages produced by the report engine.

PROPERTY	DESCRIPTION
ReportSheet.title	Sets the title meta information when generating DHTML.
ReportSheet.subject	Sets the subject meta information when generating DHTML.
ReportSheet.author	Sets the author meta information when generating DHTML.
ReportSheet.keywords	Sets the keywords meta information when generating DHTML.
ReportSheet.comments	Sets the comments meta information when generating DHTML.

ReportSheet.created	Sets the created meta information when generating DHTML.
ReportSheet.modified	Sets the modified meta information when generating DHTML.

B.3 Repository Properties

Repository properties include specifications for processing and display of reports. See the following sections for related properties in Enterprise Manager: *Changing the Server Type*, *Tailoring Server Performance*, *Configuring Server Printing*, *General Presentation*.

PROPERTY	DESCRIPTION
replet.repository.url	ServletRepository servlet URL.
replet.repository.protocol	Repository server protocol, RMI or LOCAL.
replet.indexing	Specifies, when true, that live reports should be indexed for search. See *Integration* to set this property in Enterprise Manager.
font.preload	True to pre-load the TrueType and Type 1 font information. This causes the server startup to be slower. If set to false, the loading is done the first time a PDF file is generated.
replet.cache.workset	Size (# of pages) of the working set.
replet.cache.directory	Report page cache directory. If you would like a separate cache directory for each node of a cluster, set this property as an 'Init Parameter' on each node's servlet.
replet.cache.interval	Time between repository cache cleanup.
replet.repository.file	Repository XML file path.
replet.repository.printers	A comma separated list of server printers.
replet.viewer.actions	A comma separated list of Java classes that implement user defined actions. Each action causes a button to be added to the viewer toolbar. Order is respected.
server.type	Describes the server type: rmi, servlet, etc.
repository.tree.sort	Repository tree sorting order: 'none', 'Ascend', or 'Descend'.
repository.asset.backup.enable	Enables automatic backup of the *asset.dat* file. See *Configuring the Data Space* to set this property in Enterprise Manager.
repository.asset.backup.time	Sets the time for automatic backup of the *asset.dat* file. See *Configuring the Data Space* to set this property in Enterprise Manager.
reportCache.thread.count	Maximum number of report generation threads (for 'cpu' and 'session' keys only, hard-limit). Default is 8 times the CPU count.

replet.cache.concurrency	Maximum number of report generation threads (for ‘cpu’ and ‘session’ keys only, soft-limit). Default is 4 times the CPU count.
assetData.thread.count	Maximum number of query execution threads in a Viewsheet (for ‘cpu’ and ‘session’ keys only). Default is 12 times the CPU count.
xmapTask.thread.count	Maximum number of materialized view query execution threads in a Data Node (for ‘cpu’ and ‘session’ keys only). Default is 12 times the CPU count.
repletEngine.thread.count	Maximum number of report creation threads in Repository (for ‘cpu’ and ‘session’ keys only). Default is 10 times the CPU count.
scheduleTask.thread.count	Maximum number of task execution threads in the Scheduler (for ‘cpu’ and ‘session’ keys only). Default is 6 times the CPU count.
repletService.thread.count	Maximum number of reports that can simultaneously execute queries (for ‘cpu’ and ‘session’ keys only). Increase this value if a set of reports with long-running queries is blocking the execution of other reports. Default is 12 times the CPU count.
replet.streaming	True (default) to allow the engine to deliver report pages as the data becomes available
replet.streaming.min	The number of report pages the engine should generate before streaming is initiated
replet.auto.reload	True for automatic detection and reloading of altered replet classes.
replet.auto.reload.interval	Time in milliseconds between checking for changes in replet classes.
replet.class	Sets the default class for all Template based replets, instead of ‘inetsoft.sree.TemplateReplet’
replet.custom.js	Specifies an external file containing client-side Javascript function definitions. See *Client-Side JavaScript* in *Report Scripting* for details.
replet.browseData.nocache	Caches the data when browsing column values when creating filter conditions. Defaults to false.
dependency.checker.enabled	True (default) for automatic detection and re-synchronization of report dependencies.
dependency.checker.interval	Interval between checks for changes in report dependencies, in milliseconds (default=30000)

B.4 Web Properties and Safeguards

Web properties include specifications relating to information displayed in the Portal and Portal features. See *Portal*, *Report*, and *Tailoring Server Performance* for related properties in Enterprise Manager.

PROPERTY	DESCRIPTION
graph.script.action.support	When set to ‘true’, enables interactive Viewsheet Chart features (brushing, zoom, data tip, flyover, etc.) for Charts that are modified by script. By default, these features are disabled for scripted Charts because scripts may interfere with the operation of these features.
output.null.to.zero	When set to ‘true’ (default), Viewsheet Gauges display zero (0) when an empty result set is returned. When set to ‘false’, Gauges retain the value shown for the previous non-empty result set.
javascript.java.packages	Specifies a comma-separated list of additional Java packages to be exposed in the restricted script execution environments of Visual Composer. By default, scripts created in these environments provide access only to the following Java packages: `java.awt`, `java.text`, `java.util`, `java.sql`, `inetsoft.graph.*`, `inetsoft.report`, `inetsoft.report.lens`, `inetsoft.report.filter`, `inetsoft.uql`, `inetsoft.report.painter`. Note that exposing additional Java packages may allow users to create malicious scripts, so you must thoroughly consider any such additions from a security standpoint.
replet.repository.servlet	Repository servlet URL or URI (e.g., `/sree/Reports`).
cluster.defaultProxy	Specifies a default value for an empty ‘Repository Servlet URL’ field on the ‘Server Status’ page.
html.ratio.x	Specifies the horizontal scaling ratio that relates sizes and positions of elements and fonts in the Style Studio preview to their HTML representations within the Portal. (Default=1.3.) This setting should be changed only to correct visual abnormalities in the Portal display.
html.ratio.y	Specifies the vertical scaling ratio that relates sizes and positions of elements and fonts in the Style Studio preview to their HTML representations within the Portal. (Default=1.2.) This setting should be changed only to correct visual abnormalities in the Portal display.
html.repository.header	HTML codes to insert at the beginning of the repository window. If set to ‘none’, no welcome message will be displayed above the report tree.

html.repository.footer	HTML codes to insert at the end of the repository window.
html.repository.help	Set to ‘false’ to suppress display of the ‘Help’ link below the Repository tree.
html.repository.organize	Set to ‘false’ to suppress display of the ‘Organize’ link below the Repository tree.
html.page.footer	HTML codes to insert at the end of each report page view.
html.directory	Directory used to hold HTML temporary files (images).
html.background	HTML page background color.
html.toolbar.text	Set to ‘true’ to use text-based toolbar. Default to use image buttons.
html.toolbar.location	Specifies the location of the toolbar on the HTML page: Top, Bottom, Left, Right, or Top_and_Bottom.
html.toolbar.background	Specifies the background color of the toolbar. The value is a Hex triplet, prefixed by a ‘#’.
html.toolbar.font	Specifies the foreground color of the toolbar. The value is a Hex triplet, prefixed by a ‘#’.
html.session.timeout	Inactivity timeout period for each report viewing session (in milliseconds). Default is 1200000 (20 minutes). When an open report is inactive for the specified timeout period, the report will be closed and the following message displayed: “Session timeout. Please re-open the report.”
repository.user.timeout	Inactivity timeout period for each user session (in milliseconds) for a server with session license. Default is 600000 (10 minutes). The user is logged out only when the session is inactive for the specified timeout period AND another concurrent user requires the session.
html.page.window	Determines whether a new window is created when a replet is launched from the repository view (true or false).
html.page.frame	Frame name for page view (only used if html.page.window is set to false).
html.viewer.frameset	Set to ‘true’ to use frames to display the report tree and actual reports in the same window.
html.showreplet.window	Determines whether a new window is created when a replet is launched from another report (true or false).
html.layout	This property has a value of “css” or “table”. The default is css. If generated DHTML contains overlapping elements due to some server font issues, switch to ‘table’ layout. This guarantees no overlapping of contents, but is less accurate in matching the actual layout of the report.

pdf.output.attachment	Default is true. This sends generated PDF as an attachment. If set to false, the PDF is returned as is. In IE 4 through 5, a URL referring to dynamically generated PDF could result in a blank page. Optionally, it can be set to 'embed', which will cause a PDF request to return a HTML page with an <embed> tag, which normally displays correctly in IE without causing a dialog to prompt users to save the PDF.
html.retry.interval	Milliseconds between retry, when waiting for a report.
html.session.max.report	The maximum number of reports that can remain active in a particular session. Default is 5.
html.browser	Client-side HTML browser full path.
html.encoding.utf8	Default is true. If set to false, the HTML output is not encoded using UTF8. This may cause international characters to be displayed improperly.
html.offset.x	The distance of the generated HTML contents to the left edge of the browser.
html.offset.y	The distance from the generated HTML contents to the top of the browser.
html.export.button	Set to false to remove the 'export' button from the report toolbar.
html.export.embed	Set to true to embed the output of the export (from the export dialog) into the report frame and replace the report contents.
html.find.button	Set to false to remove the 'find' button from the report toolbar.
html.findNext.button	Set to false to remove the 'find next' button from the report toolbar.
html.mail.button	Set to false to remove the 'mail' button from the report toolbar.
html.pdf.button	Set to false to remove the 'pdf' button from the report toolbar.
html.page.button	Set to false to remove 'Page ____ of ____'.
html.refresh.button	Set to false to remove the 'refresh' button from the report toolbar.
html.serverPrint.button	Set to false to remove the 'server print' button from the report toolbar.
html.toc.button	Set to false to remove the 'table of contents' button from the report toolbar.
html.customize.button	Set to false to remove the 'customize' button from the report toolbar.
html.archive.button	Set to false to remove the 'save in archive' button from the report toolbar.
html.compose.button	Set to false to remove the 'edit and analyze' button from the report toolbar.

html.defaultPrint.button	Set to false to remove the 'Default Print' button from the report toolbar. See *Report Toolbar Options* to set this property from Enterprise Manager.
html.repservlet.title	The Portal page title. If unspecified, the default title of 'Style Report Viewer' will be used.
html.tblheading.hide	Set to true to include table and section grouping headers in the TOC.
dhtml.error.page	Defines the replet error page.
export.menu.options	A comma separated list of option numbers. Only the options on the list are added to the report export menu. e.g: PDF, EXCEL, EXCEL_SHEET, EXCEL_DATA, RTF, RTF_LAYOUT, HTML, HTML_NO_PAGINATION, HTML_BUNDLE, HTML_BUNDLE_NO_PAGINATION, CSV, SVG, TEXT, REPORT
excel.export.grid.show	When true, specifies that gridlines will be shown in the Excel export, and that default text backgrounds are transparent. When false, gridlines are suppressed and default text backgrounds are white.
rtf.image.scale	Boolean that determines whether the image is exported in its original size in RTF format.
rtf.hyperlink.indicator	Set to false to remove hyperlink underlining in RTF format.
repository.buttonset	The resource path to the directory that contains the icons for toolbar buttons. The following images should be found in this directory: – archive.gif – composer.gif – customize.gif – export.gif – find.gif – firstpage.gif – lastpage.gif – mail.gif – menuarrow.gif – nextpage.gif – pdf.gif – prevpage.gif – refresh.gif – sprinter.gif – toc.gif
query.preview.maxrow	Sets the maximum number of rows for queries in Worksheet and Viewsheet live data mode.
query.preview.timeout	Sets the timeout interval (seconds) for queries in Worksheet and Viewsheet live data mode.
csv.import.max	Sets the maximum size in bytes for CSV files imported into a Worksheet Embedded Table (see *Importing Data Into an Embedded Table* in *Advanced Data Mashup*).

query.runtime.maxrow	Sets the global maximum size for all queries during report/Viewsheet runtime execution.
table.output.maxrow	Sets the global maximum size for all tables during report/Viewsheet runtime execution.
report.output.maxpages	Sets the global maximum on the number of pages for reports during runtime execution.
query.runtime.timeout	Sets the global timeout (seconds) for all queries during report/Viewsheet runtime execution.
asset.sample.maxrows	The default 'Design mode sample data size' in the Worksheet Options dialog box. See *Setting Global Worksheet Options* in the *Advanced Data Mashup*.
DistinctTableLens.maxrow	Sets the maximum number of distinct values to display in the Browse Data menu.
table.export.format	When set to "`csv`", table data exported from a Viewsheet or Data Worksheet will use CSV format. Some formatting may be lost. When set to "`xls`", exported data will use Excel format, spanning multiple Excel sheets if needed, and preserving all formatting. If the property is left unspecified, tables with greater than 10000 rows are exported in CSV format, otherwise in Excel format.
query.estimate.enabled	When true (default), the product asks the database engine to estimate the duration of executing worksheet queries. If the returned value is over a certain threshold, the query does not run.
asset.ignore.filtering	When true (default), this displays a warning message to the Viewsheet designer when a Worksheet Data Block returns no data. The warning indicates that conditions on *all* Data Blocks will be dropped in Viewsheet design view (only) to help facilitate formatting. When false, all conditions are kept in place (which may make Viewsheet formatting more difficult).
assetData.thread.count	The maximum number of Viewsheets that can simultaneously execute queries. Increase this value if a set of Viewsheets with long-running queries is blocking the execution of other Viewsheets. (Default is six times the number of CPUs.)
graph.ggraph.maxcount	The maximum number of nested graphs for Viewsheets and reports (Facet Chart). An error is generated if this value is exceeded.
replet.myreports.disabled	Set to true to hide the 'My Reports' folder from the replet tree.
enable.changePassword	Set to false to disable users' ability to change passwords in the User Portal (using the default security provider).

em.user.max	For LDAP security, the maximum number of users that will be listed in the user tree (under the **User** tab or **Security** tabs). The default is 1000. To find unlisted users, see *Searching the User/ Role Tree*.
mv.repository.servlet	The complete URL of the Data Server machine, which is required for scheduling materialized views.
mv.detail.data	When set to true, populates Viewsheet Chart and Crosstab 'Show Details' data from an existing materialized view rather than from the database.
mv.cache.data	When set to true (default), caches materialized view result sets so that they can be shared across Viewsheets.
mv.backup.dup	The number of successive materialized view versions to maintain as backups. The default is 1. See *Managing Materialized Views* for more information.
hyperlink.indicator	When set to true, automatically displays hyperlinks with underlining. When set to false (default), displays hyperlinks without underlining. (Users can manually format the hyperlinks as desired.)
viewsheet.chart.adhoc	When set to true (default), configures the 'Enable Ad Hoc Editing' property of Viewsheet charts to be enabled by default.
viewsheet.default.font	Specifies the default font for all Viewsheets.
viewsheet.font.size	Specifies the default font size for all Viewsheets. A plain integer value specifies an absolute size for all fonts. An integer preceded by a + or - indicates an increase or decrease in size from the default font sizes (e.g., `viewsheet.font.size=+5`).

B.5 Security Service Properties

Security properties include specifications relating to security provider. See *Specifying a Security Provider* and *Tailoring Server Performance* for related properties in Enterprise Manager.

PROPERTY	DESCRIPTION
security.password.file	Password file used by the default security service.
security.acl.file	Access the control file used by the default security service.
security.use.cookie	Set to false to disable the use of cookies in security service.
security.ldap.userbase	The base DN for users. For iPlanet, default is 'ou=People', and for ADS, default is 'cn=Users'.

security.ldap.groupbase	The base DN for groups. For iPlanet, default is ‘ou=Groups’, and for ADS, default is ‘cn=Users’.
security.ldap.rolebase	The base DN for roles. For iPlanet, default is ‘ou=People’, and for ADS, default is ‘cn=Users’.
security.ldap.subtree	Set to ‘true’ to enable recursive search of specified LDAP user base. Default is ‘false’.
security.cache	Caches data when using LDAP security for improved performance.
security.allow.iframe	Set to ‘false’ to prevent the Enterprise Manager and User Portal login pages from being displayed within an IFrame. This is equivalent to setting the HTTP header property X-Frame-Options to DENY, and should be used in conjunction with Single Sign-On (SSO). See *Configuring Single Sign-On (SSO)* in *Advanced Server Integration* for more information. The default is ‘true’.
role.administrator	Only users with this role can log into the EM and administer the report server. For an LDAP/AD security provider it maps to a Security Group. It defaults to ‘Administrator.’ e.g., role.administrator=SRAdmin

B.6 Scheduling Service Properties

Scheduling properties include specifications relating to the RMI Scheduler application. See *Configuring the Scheduler* and *Scheduler Properties* for related properties in Enterprise Manager.

PROPERTY	DESCRIPTION
schedule.task.file	Schedule a XML file path.
schedule.task.listener	Specifies the fully-qualified name of a listener class to be executed at the start and end of every scheduled task.
schedule.cycle.interval	Interval between batch cycles (in milliseconds).
schedule.reload.auto	Set to true to automatically reload schedule file when it is modified.
schedule.reload.interval	Interval (in milliseconds) for checking schedule file changes. Default is 2000.
scheduler.classpath	The path that will be prepended to the system classpath when the Scheduler process is started.
schedule.log.file	Scheduler log file path.
schedule.memory.min	Minimum memory usage for the Scheduler.

schedule.memory.max	Maximum memory usage for the Scheduler.
schedule.abnormity.notify	Specifies a comma-separated list of emails to receive notification in the event that the Scheduler application becomes unresponsive.

B.7 Mailing Service Properties

Mail properties include specifications relating to email notification and report distribution. See *Configuring Server Email* for related properties in Enterprise Manager.

PROPERTY	DESCRIPTION
mail.split.message	Message used when a PDF file is split into multiple parts.
mail.subject.format	Format for the mail subject line. It is a message format (java.text.MessageFormat). There are two parameters used in generating the subject line. The first parameter is the alias of the replet, and the second parameter is the date and time (Date) of the report generation. For example, a possible format could be "Report {0} generated at {1,time} on {1,date}."
mail.smtp.host	Mail SMTP host name.
mail.tls	When set to 'true', specifies that a STARTTLS command should be issued. The default is 'false'.
mail.smtp.port	Specifies the SMTP port number. The default for non-SSL use is port 25. The default for SSL use is port 465.
mail.attachment.max	Maximum size for a PDF attachment on an email. If the PDF file is larger than the maximum size, the PDF is split into multiple parts.
mail.from.address	'From' email address used in email message.
system.admin.address	Email address of system administrator for the replet error page.

B.8 Log Service Properties

Please see section 'Advanced Log Configuration'. For customizing log detail levels and log message formats.

PROPERTY	DESCRIPTION
log.output.stderr	True to send output to the stderr.
log.output.file	Log file path.
log.detail.level	Number specifying the logging level, as follows: **Finest** (fine-grained performance information), **Fine** (debugging messages), **Info** (non-critical information messages), **Warning** (non-fatal errors and critical messages), **Severe** (fatal errors).

log.view.maximum	The maximum number of lines to return when the ‘All’ option is selected on the ‘Log View’ page.
log.view.lines	The number of lines to display on the log page.

B.9 PDF and Font Properties

PDF properties include specifications relating to reports generated in PDF format. See *PDF Generation* for related properties in Enterprise Manager.

PROPERTY	DESCRIPTION
pdf.output.ascii	True to generate only ASCII output in the PDF file. No ASCII data is encoded to seven bit data.
pdf.compress.image	True to compress all image data.
pdf.compress.text	True to compress all text data.
pdf.map.symbols	Map symbol characters to symbol fonts. This is required if symbol characters are used in reports.
pdf.font.mapping	Font mapping from Java font to fonts used in PDF. It is a semi-colon separated pairs of font names. Each pair is separated by a colon, e.g., ‘MS Hei:HeiseiKakuGo-W5-Acro;MS Song:Song-Acro’. See also *CJK Fonts*.
pdf.print.scaling	True to make the PDF reader use the application default for the ‘Page Scaling’ print option. False to use ‘None’
pdf.embed.font	True to embed the font in the generated PDF file.
pdf.embed.cmap	True to embed the cmap in the generated PDF file.
pdf.open.bookmark	True to have bookmarks immediately displayed when the generated PDF files are opened in Adobe Acrobat.
pdf.open.thumbnail	True to have thumbnails immediately displayed when the generated PDF files are opened in Adobe Acrobat.
pdf.generate.links	True to create hyperlinks in the resulting PDF file. False to turn off hyperlink generation and speed up the pdf generation process.
font.truetype.path	TrueType font directories.
font.metrics.source	TrueType to use TrueType fonts as specified in the directories in font.truetype.path.
font.afm.path	Type 1 AFM file directories.
font.cmap.path	Cmap file directories. Only used when processing CJK characters.

font.ratio.x	A list of ratios used to adjust horizontal character spacing. This is mostly used when the font width information is different between Java and PDF. The ratio can be used to adjust the spacing so that the generated PDF matches the actual font size used in PDF viewers. It is a list of semi-colon separated ratios. Each ratio is specified by a font name and, optionally, a font style, followed by the ratio, e.g., MS Hei:1.1;Algerian-bolditalic:1.02.
font.ratio.y	Similar to font.ratio.x, but is applied to the font vertical spacing.

B.10 Image Properties

Image properties include specifications relating to the representation of images in the Portal. See *General Presentation* for related properties in Enterprise Manager.

PROPERTY	DESCRIPTION
image.type	PNG or JPEG. PNG can only be used with version 4.0 or newer browsers. Generally PNG results in an image with a smaller file size, and better quality than JPEG.
image.filtered	Default is false (recommended). Turns PNG compression on or off. This property is ignored if JPEG images are used.
image.antialias	PNG images are antialiased by default when using j2d. Set this property to false to turn off antialias. (Antialias gets rid of “jaggies”.
html.image.xratio	When generating html, images are converted to jpeg using a resizing ratio of 1.0. This sometimes causes the image to be distorted. This default can be changed by setting this property.
html.image.yratio	When generating html, images are converted to jpeg using a resizing ratio of 1.0. This sometimes causes the image to be distorted. This default can be changed by setting this property.
html.image.scale	Set to ‘true’ to scale images using x/y ratio; otherwise, natural size is used, regardless of the size setting on the report.
html.image.count	Specifies the maximum number of images cached on the report server. Reducing the number of cached images can help alleviate certain server memory issues.

B.11 Search Properties

Search properties include specifications for searching within reports.

PROPERTY	DESCRIPTION
search.pagelocation.perpage	Specify the number of search results displayed per page. The default value is 30.
search.pagelocation.brieflength	Specify the length of the description for each search result. The default value is 45.

B.12 Archive Properties

Archive properties include specifications relating to archived (static) reports saved to the repository. See *Archive* for related properties in Enterprise Manager.

PROPERTY	DESCRIPTION
report.archive.CVSPath	The path used to look for cvs executables when using CVS based archiving.
report.archive.storage	Report archive's storage option. Valid options are: No Archive, CVS, File System, Custom Archive
report.archive.compress	Set to 'true' to compress the files saved in the report archive. Only the files that are not already compressed will be compressed, e.g., .sro files and html bundles. Compressed files can be accessed as normal from the viewer with the same file name as before. However, the compressed files will be saved in the underlying storage as *filename.zip*.

B.13 Miscellaneous Properties

The following table lists a variety of other properties.

PROPERTY	DESCRIPTION
mysql.server.timezone	Set to a valid Java time zone ID to indicate the MySQL database server time zone (e.g., "US/Eastern").
local.timezone	Set to a valid Java time zone ID to indicate the Style Intelligence server time zone (e.g., "US/Eastern"). If not defined, the default time zone is used.
accessibility.enabled	Set to 'true' to enable features for Rehabilitation Act Section 508 compliance.
format.time	Format used for all time data (Date). See *Formatting* in the *Advanced Report Design* for more information.
format.date	Format used for all date data. See *Formatting* in the *Advanced Report Design* for more information.

format.number.round	If a number format is not specified in the designer, this property value controls the rounding of the numbers. The property values are the same as the valid arguments for the '`formatNumber()`' method. See *Global Functions* for more details.
string.compare. caseSensitive	Default grouping and ordering is now case-insensitive, which is consistent with most databases. Set this property to true to revert to pre-9.1 behavior.
export.csv.delimiter	Delimiter used when exporting a CSV file.
export.csv.quote	Quote character used when exporting a CSV file.
export.text.table. maxRowAlign	Set the number of rows to look at when determining the column width to use in Text exports.
export.Excel.isHeader FooterInPageSetup	If "true," top and bottom margins, and headers and footers will be set in Excel's "Page Setup"
excel.close.factor	For Excel export, a factor determining whether nearby report elements are placed in adjacent cells or spaced apart. A smaller factor creates more space between elements in the exported spreadsheet.
excel.colwidth.max	For export to Excel (best data editing), specifies the maximum permitted Excel column width. If the report element width exceeds the maximum, it is represented in Excel by merging multiple columns with widths given by excel.colwidth.normal.
excel.colwidth.normal	For export to Excel (best data editing), specifies the column width to use for merged columns when a report element's width exceeds excel.colwidth.max. A value of -1 indicates that no merging should be performed.
excel.rowheight.max	For export to Excel (best data editing), specifies the maximum permitted Excel row height. If the report element height exceeds the maximum, it is represented in Excel by merging multiple rows with heights given by excel.rowheight.normal.
excel.rowheight.normal	For export to Excel (best data editing), specifies the row height to use for merged rows when a report element's height exceeds excel.rowheight.max. A value of -1 indicates that no merging should be performed.
excel.font.ratio	Sets the proportion by which font size is globally reduced for Excel exports. (Font size reduction can often improve the appearance of Excel exports.) For example, setting excel.font.ratio=0.85 uniformly reduces font sizes within the exported report to 85% of their size in the original report. To set this proportion on a font-by-font basis, use the property template excel.font.<*fontname*>.ratio, e.g., excel.font.arial.ratio=0.80.

query.variable.unique	If query variables are declared as unique, variables with the same name across different queries are treated as one variable. Otherwise, each query contains its own set of variables.
exceed.session.page	Contains a path to an html file that contains a message to display when the maximum number of concurrent users has been exceeded. If this is not set, the default message is used.
jdbc.connection.pool	Class Name for a user defined connection pooling class. This class should implement inetsoft.uql.jdbc.ConnectionPool
jdbc.connection.pool.size	Initial size of the connection pool.
jdbc.connection.pool.maxsize	Max size to which the connection pool will grow, if required. Default is 15.
jdbc.max.cursor	Maximum number of cursors allowed per database connection. It may be necessary to set this property to 1 if an Oracle database is observed to execute queries sequentially rather than concurrently. Default for Oracle is 300, and for other databases is 1.
db.caseSensitive	Indicates (when set to true) that database column names have been defined in the database software as case-sensitive.
sree.bundle	The name of the report localization bundle.
index.directory	The directory where this software stores indexed data that is used for searching the archive.
fiscal.start.month	The month in which the fiscal year begins.
fiscal.start.day	The day in the fiscal.start.month on which the fiscal year begins.
default.browser	This defines the default browser used on Unix machines. It defaults to Mozilla.
barcode.bar.width	The width (in pixels) of the thinnest bar in a barcode. Other bars will change their size relative to this.
barcode.resolution	The output resolution of the barcode when either the barcode is being outputted to a device other than the screen, or the barcode is being generated on a headless machine and the screen resolution could not be determined. The default resolution is 72 dpi.
text.encoding.utf8	By default it is set to true. If this property is false, you will use the encoding specified for the local operating system instead of the UTF8 encoding.
map.shp.dir	Property to specify the absolute path of the folder which contains the .shp and .jar files used by the map element.
script.execution.timeout	A safety-net for script execution time, especially for infinite loops. Times-out the script if execution time exceeds specified value (in seconds).

olap.table.originalContent	Set to ‘true’ (default) to display the value of a CUBE dimension in a Selection List with all its hierarchical components, otherwise set to ‘false. (Applies to Chart labels and Selection Lists.)
olap.drillthrough.maxrows	Determines how many rows to display for a Show Details operation against an OLAP cube. Default is 10.
olap.table.maxrow	Determines how many rows to retrieve when creating unique values for selection lists. Default is 10000.
olap.cancel.enabled	Set to ‘true’ (default) to enable query cancellation for OLAP databases. This feature cancels currently executing queries when a user makes updates to their selections. Set to ‘false’ to prevent errors on databases that do not support query cancellation.
anonymous.userdata.save	Set to ‘false’ to prevent storage of preferences for the anonymous user.
login.loginAs	Set to ‘on’ to permit an administrator to log in as a particular user. (Adds a ‘Login as’ field to the Portal login screen.) See *Emulating a User Login*.
remove.outerJoin.only	When ‘false’, the query engine removes joined tables with no columns in the ‘select’ and ‘where’ clauses, such as when a condition is dropped due to null parameter.
vs.font.asDefault	When set to true, uses the Viewsheet default font for a Table rather than the Table Style font. When set to false (default), uses the specified Table Style font.
sort.crosstab.aggregate	When set to true, allows a Crosstab in a Viewsheet to be sorted by aggregate value. Default is `false`.
sort.crosstab.dimension	When set to true (default), allows a Crosstab in a Viewsheet to be sorted by dimension value.

APPENDIX C: Deploying the Application as a WAR File

The default Style Intelligence installation includes a Jetty web server. To deploy the Style Intelligence application to a different web server, you should create a WAR file from the installed version. You can deploy the WAR file to any web server that supports the Servlet 2.2 specification and allows WAR files to be used for deployment.

General information on creating and deploying WAR files is available at the following sites:

- http://docs.oracle.com/javaee/5/tutorial/doc/bnadx.html
- http://docs.oracle.com/cd/E13222_01/wls/docs70/webapp/basics.html#100368
- http://documentation.progress.com/output/Iona/orbix/6.1/tutorials/fnb/dev_intro/j2ee_overview8.html

The following sections provide some helpful information for deploying a WAR file on some common servers. Please refer to your server documentation for more detailed instructions.

You only need to do the WAR file deployment once. After the application has been initially deployed, you can deploy additional assets (data sources, reports, dashboards, etc.) using *Incremental Deployment of Assets*.

C.1 Creating a WAR File

A *WAR file* is a regular JAR file with a special directory structure. The root directory of the archive file serves as the document root for serving files that are part of the application. A special directory, *WEB-INF*, contains all servlet JAR files, class files, and resources.

```
/index.html
/logo.gif
/WEB-INF/lib/{JAR files used by servlet}
/WEB-INF/classes/{Class files used by servlet}
...
```

Mixing JAR files and class files may be problematic in some servlet runtime implementations. Therefore, we recommend packaging all class files into JAR files, and then placing them in the */WEB-INF/lib* directory.

Example: WAR File Creation

1. Create a root directory to hold the files. In this example, you create a directory in *c:\temp* called ‘sree’.

2. Create the following sub-directories in *c:\temp\sree*:

```
WEB-INF\lib
WEB-INF\classes
```

3. Create an HTML entry screen in the root directory. You can use the default login screen; simply copy the 'index.html' and 'logo.gif' files from the existing installation's *webapps\sree* and *webapps\sree\images* directories, respectively, and paste these files into the new *c:\temp\sree* directory.

4. Copy the SREE runtime JAR files to the *c:\temp\sree\WEB-INF\lib* directory. The required JAR files are:

```
sree.jar, or one of the following analogs:
   bisuite.jar
   visual.jar
   sree.jar
etools.jar
```

5. Create a JAR file to contain all of the replet class files in your report application. The JAR file should also contain all resource files, such as the report template `.srt` files and data files. For example, to create a JAR file for all of the reports in a given directory, navigate to the desired directory, and then run the following command:

```
jar cf guide.jar replets\*.class replets\*.srt replets\*.txt
```

6. Copy the resulting JAR file *guide.jar* to *c:\temp\sree\WEB-INF\lib*.

7. Copy the configuration file *sree.properties* to *c:\temp\sree\WEB-INF\classes*.

8. Copy the *repository.xml* file to *c:\temp\sree\WEB-INF\classes*. The file name may be different depending on the property value of `replet.repository.file` in *sree.properties*.

9. If your application uses the default security implementation of SREE, copy both the *acl.xml* and *passwd.xml* files to *c:\temp\sree\WEB-INF\classes*. The file names may be different depending on the property values of `security.password.file` and `security.acl.file`.

10. Create a *web.xml* file in the *WEB-INF* directory. The *web.xml* file should be in standard web deployment descriptor format. The following is an example of the descriptor file:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD
Web Application 2.3//EN" "http://java.sun.com/dtd/web-
app_2_3.dtd">
<web-app>
```

```
<servlet>
   <servlet-name>replets</servlet-name>
   <servlet-class>inetsoft.sree.web.ServletRepository</
   servlet-class>
</servlet>
<servlet>
   <servlet-name>manager</servlet-name>
   <servlet-class>inetsoft.sree.adm.AdmServlet</servlet-
   class>
</servlet>
<servlet>
   <servlet-name>DataServlet</servlet-name>
   <servlet-class>DataServlet</servlet-class>
</servlet>

<servlet-mapping>
   <servlet-name>replets</servlet-name>
   <url-pattern>/Reports</url-pattern>
</servlet-mapping>
<servlet-mapping>
   <servlet-name>manager</servlet-name>
   <url-pattern>/EnterpriseManager</url-pattern>
</servlet-mapping>
<servlet-mapping>
   <servlet-name>DataServlet</servlet-name>
   <url-pattern>/DataServlet</url-pattern>
</servlet-mapping>
</web-app>
```

11. Create the WAR file as follows:

```
cd c:\temp\sree
jar cf guide.war index.html logo.gif WEB-INF
```

These steps can be placed in a shell script or a makefile, and can then be performed as part of the application building process.

Once you have created a WAR file, deploy it in a servlet runtime environment. The procedure for deploying a WAR file is slightly different in different implementations. Instructions for some typical environments appear in the next few sections. Please refer to your server documentation for more detailed instructions.

¢

C.2 WebSphere 5.0 and higher

Follow the steps below to install the application on a WebSphere server. You must supply the application WAR file to deploy.

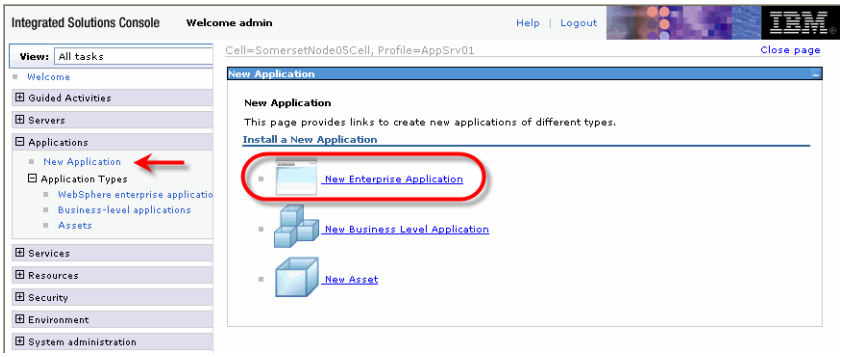
1. Install WebSphere by following the vendor's instructions.

2. If you have not already done so, create a WAR file containing the reports you wish to deploy. See Appendix C.1, *Creating a WAR File*, for information on how to create a WAR file manually.

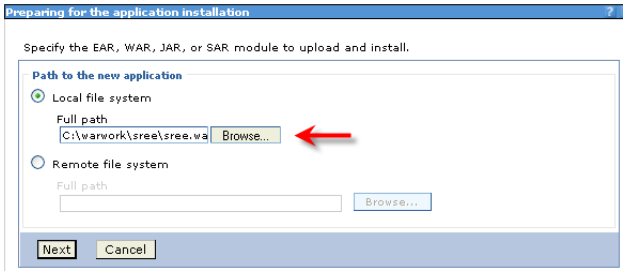
3. Run the WebSphere Administrative Console (default, `http://localhost:9090/admin`) and login.

4. On the left side of the screen, expand the 'Applications' option, and click on 'New Application'.

5. On the 'New Application' page, select 'New Enterprise Application'.

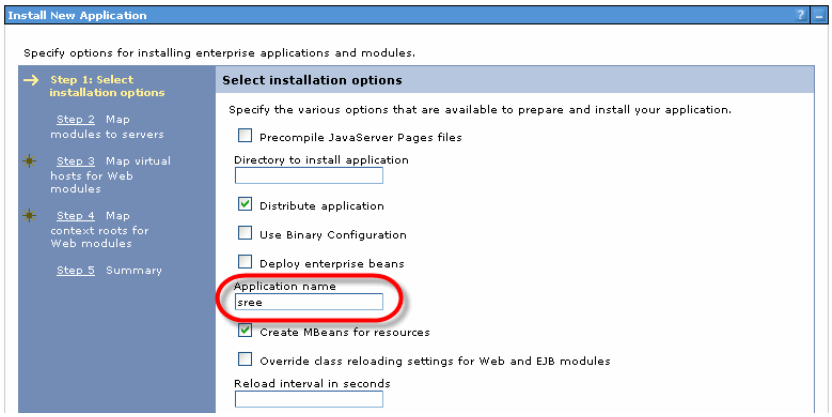


6. In the 'Path to New Application' panel, select the 'Local File System' option, and type the location of the WAR file in the 'Full Path' field (*C:\warwork\sree\sree.war* in the example below), or use the 'Browse' feature to locate the file. Click 'Next'.



7. Under 'How do you want to install the application?', select 'Fast Path'. Click 'Next'.

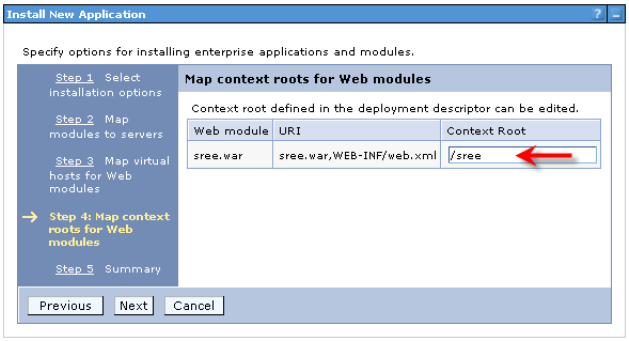
8. On the 'Select installation options' page, set the 'Application name' to match the WAR file name. (In the example, the 'Application name' is '`sree`' because the WAR file is '`sree.war`'). Click 'Next'.



9. On the 'Map modules to servers' page, select the '`sree.war`' module. Click 'Next'.

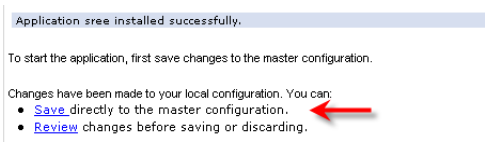
10. On the 'Map virtual hosts for Web modules' page, select the '`sree.war`' module. Click 'Next'.

11. On the 'Map context roots for Web modules' page, enter '`/sree`' for the 'Context Root'. This should be the same as the WAR file name. (In the example below, the 'Context Root' is '`/sree`' because the WAR file is '`sree.war`'.) Click 'Next'.



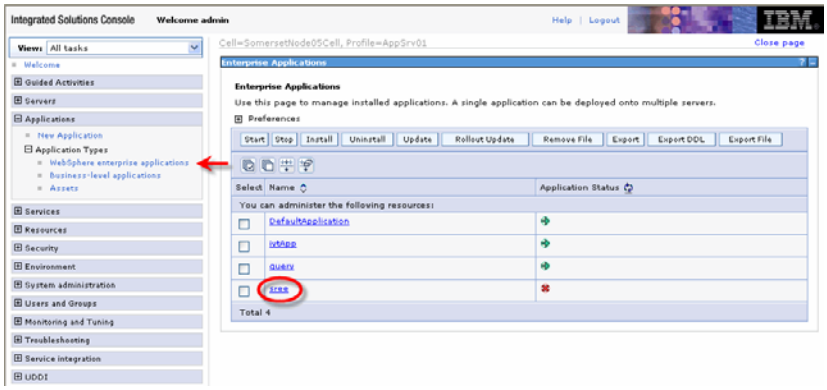
12. On the 'Summary' page, click 'Finish'. Wait until the application installs successfully.

13. Press 'Save directly to the master configuration'.

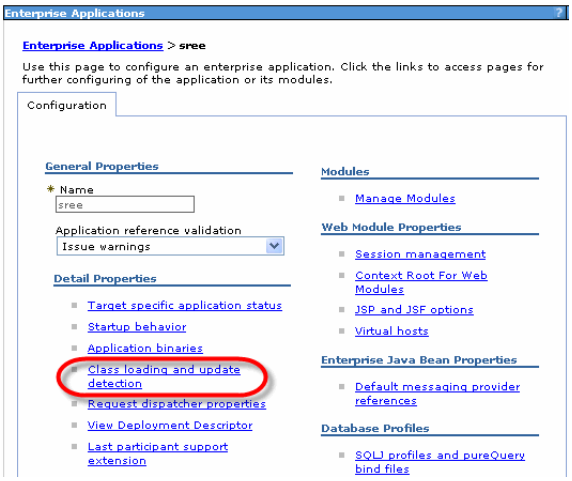


14. When the master repository is updated, expand the 'Application Types' node under 'Applications' (in the left panel), and click 'Websphere enterprise applications' link.

15. On the ‘Enterprise Applications’ page, click the link corresponding to the application that you just created (`sree`).



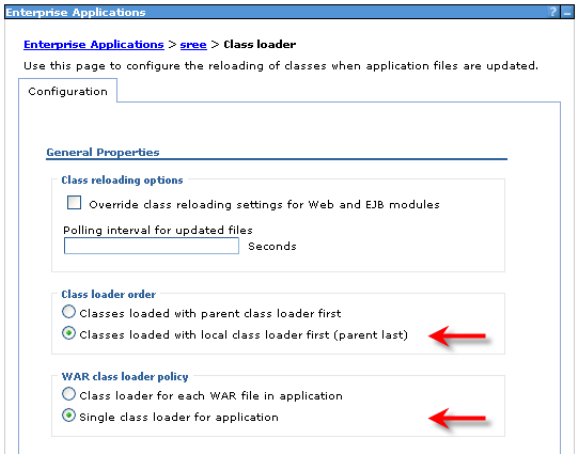
16. In the ‘Configuration’ panel, click the ‘Class loading and update detection’ link.



Change the following settings:

a. In the ‘Class loader order’ panel, select ‘Classes loaded with local class loader first (parent last)’.

b. In the ‘WAR class loader policy’ panel, select ‘Single class loader for application’.



c. Click ‘OK’.

17. When prompted, click ‘Save directly to the master configuration’.

18. In the ‘Enterprise Applications’ list, select the new application (`sree`), and click the ‘Start’ button.

Once you have deployed the application, you can launch the User Portal, Enterprise Manager, and Soap Repository from the following URLs, respectively:

```
http://localhost:9080/sree/Reports

http://localhost:9080/sree/EnterpriseManager

http://localhost:9080/sree/SoapRepository?wsdl
```

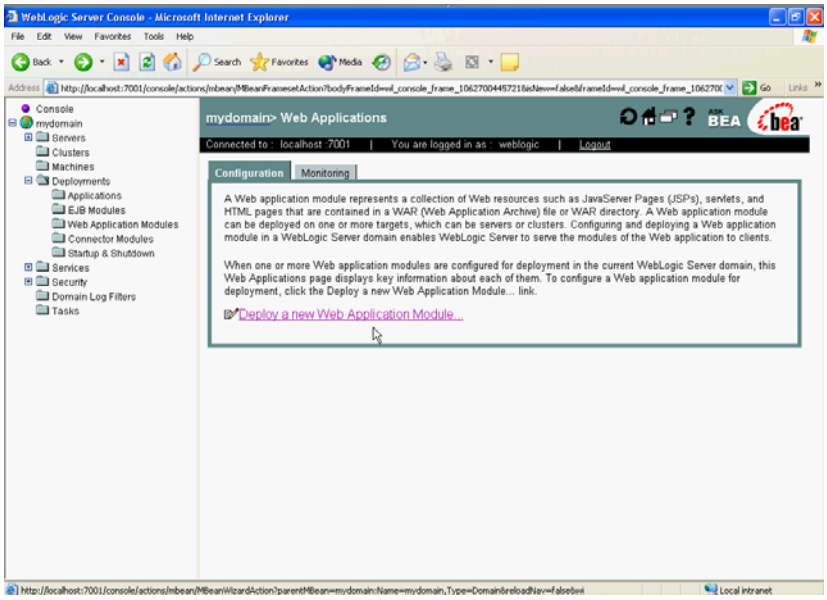
C.3 WebLogic 7.0 and Higher

Create the WAR file you wish to be deployed. See Appendix C.1, *Creating a WAR File* for information on how to create a WAR file manually. To deploy the application, follow the steps below. (Assume the WAR file you want to deploy is named *c:\temp\sree\sree.war*.)

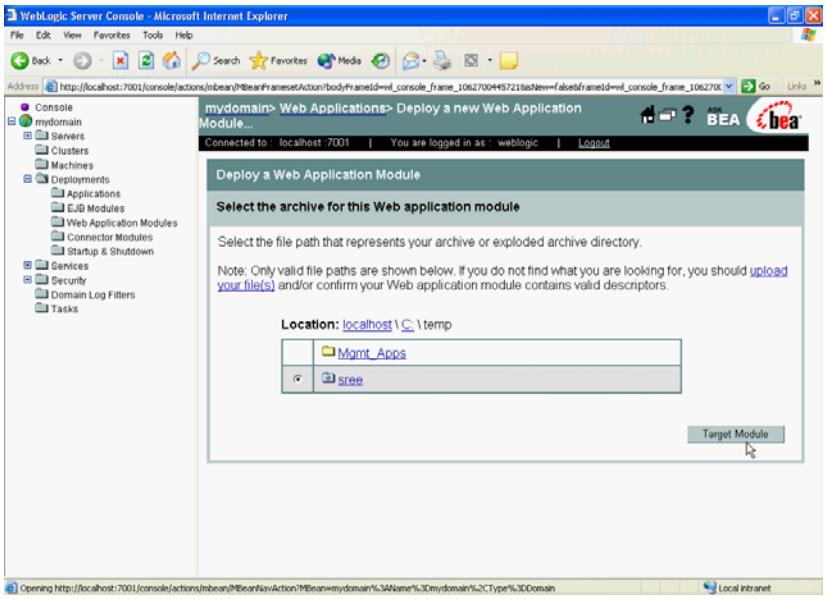
1. Start the default server and then the administrative console.

2. Expand the tree in the left pane and expand ‘Deployments’.

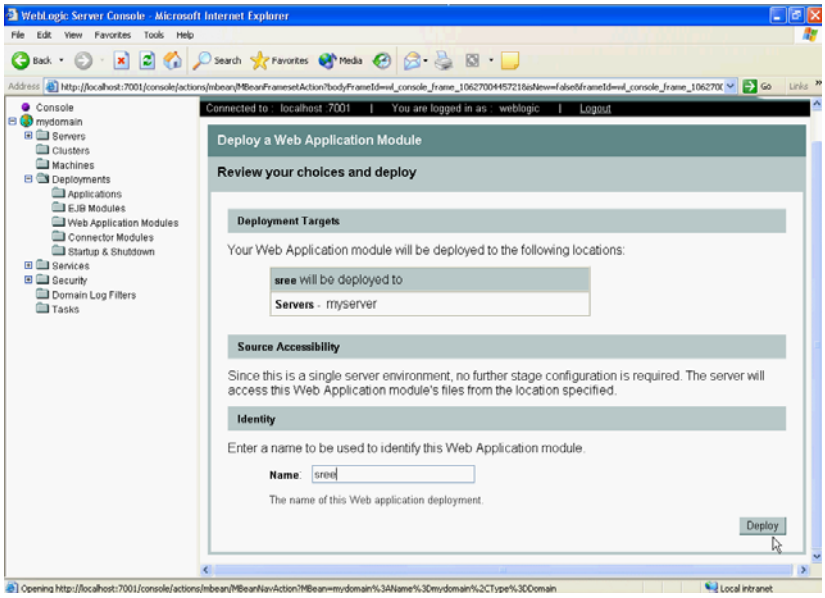
3. Click on ‘Web Application Modules’ and in the right pane click on ‘Deploy a new Web Application Module’.



4. Navigate through the directory structure to find and select the WAR file, and click on ‘Target Module’.



5. Review your choices and deploy the module (clicking on ‘Deploy’ in the lower-right corner).



6. The new application is now deployed within WebLogic.

7. Start up a browser. You should be able to access your servlet at '`http://localhost:7001/{WAR}/{Serv}`', where `{WAR}` is the name of the WAR file you installed and `{Serv}` is the name of the servlet.

C.4 Tomcat 4.1 and Higher

Resources for installing and configuring Tomcat can be found at http://jakarta.apache.org/tomcat/index.html.

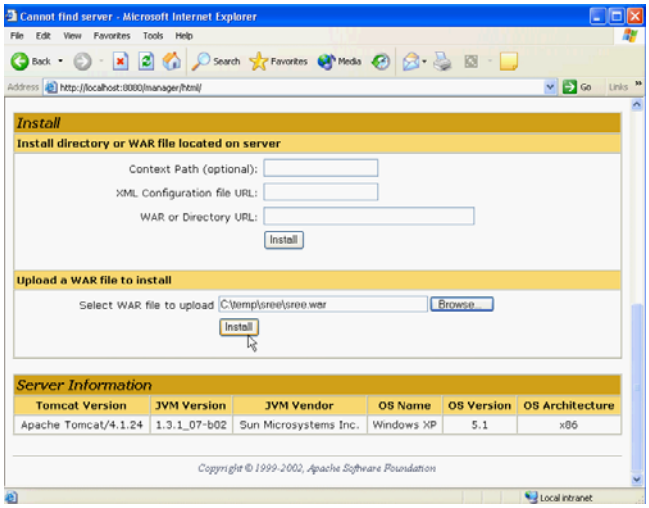
To deploy the Style Intelligence application to Tomcat, you can simply copy the entire *sree* directory from the installed version (without creating a WAR file) into the Tomcat server. Make sure that any paths specified in the *sree.proprties* file are correct for the new installation.

If you want to deploy the application using a WAR file, first create the WAR file as described in Appendix C.1, *Creating a WAR File*. Then deploy by copying the WAR file to the *<CATALINA_HOME>/webapps* directory, where *<CATALINA_HOME>* is the directory into which Tomcat 4.1 has been installed. (Exception: *<CATALINA_BASE>/webapps* should be used if you have used '*CATALINA_BASE*' to configure Tomcat for multiple instances). The Tomcat server will automatically expand and deploy the WAR file when it starts up.

Alternatively, you can deploy the WAR file using the Tomcat Web Application Manager, as follows:

1. Load '*http://localhost:8080/manager/html/*' in a browser to access the Tomcat Web Application Manager. (You will need to log in as a Tomcat user that has a 'manager' role.)

2. Scroll down to the bottom of the page and use the 'Upload a WAR file to Install' option to install your WAR file.



For either of the above approaches, the User Portal can be launched from the following URL:

```
http://localhost:8080/sree/Reports
```

Likewise, Enterprise Manager can be launched from the following URL:

```
http://localhost:8080/sree/EnterpriseManager
```

APPENDIX D: Modifying Data Source Definition During Deployment

This section explains how to use the built-in Ant task to automatically change the data source definition.

D.1 Using the Ant Task to Change Data Source

To use the built-in Ant task to automatically change the data source definition, follow the steps below:

1. Make sure Ant-related JAR files and *sree.jar* (or analogs *visual.jar*, *bisuite.jar*) are included on the classpath.

2. Create an Ant task definition in `build.xml` as follows:

   ```
   <taskdef name="dsm"
   classname="inetsoft.uql.util.dsm.DataSourceModifier"/>
   ```

3. Within an Ant target, include the `<dsm>` tag as follows:

   ```
   <dsm restore="true" dsfile="datasource.xml">
   </dsm>
   ```

   where:
   '`restore`' is set to true to restore data source file after deployment
   '`dsfile`' specifies the data source file's absolute path

   Thus, use the dsm task to modify the `datasource.xml` and use the WAR or JAR task to create a webapp. If `restore=true`, the `datasource.xml` file will be restored back to its original state after the build is finished.

4. Depending on the type of database, within the `<dsm>` tag, define one of the attributes listed under *Available DSM Tags*.

5. Run the Ant task.

D.2 Available DSM Tags

This section lists the available attributes for the `<dsm>` tag. You do not need to specify *all* the attributes listed here, only the ones that you wish to modify.

<ds_jdbc>

This tag is to be used when the data source is of type 'JDBC'.

```
<ds_jdbc name="DB2" url="jdbc_url" driver="jdbc_driver"
 requireLogin="true" user="jdbc_user"
 password="jdbc_password"
 defaultDB="jdbc_defaultDB"/>
```

– ‘name’ is the jdbc data source’s name
– ‘url’ is the jdbc data source’s new url
– ‘driver’ is the jdbc data source’s new driver
– ‘requireLogin’ is true if the jdbc data source requires login
– ‘user’ is the jdbc data source’s new user if requires login
– ‘password’ is the jdbc data source’s new password if requires login
– ‘defaultDB’ is the jdbc data source’s new defaultDB

<ds_text>

This tag is to be used when the data source is of type ‘Text’.

```
<ds_text name="chart" url="text_url" method="get"/>
```

– ‘name’ is the text data source’s name
– ‘url’ is the text data source’s new url
– ‘method’ is the text data source’s new method: ["get"|"post"|""]

<ds_corba>

This tag is to be used when the data source is of type ‘Corba’.

```
<ds_corba name="Bank" host="corba_host" port="corba_port"/>
```

– ‘name’ is the corba data source’s name
– ‘host’ is the corba data source’s new host
– ‘port’ is the corba data source’s new port

<ds_ejb>

This tag is to be used when the data source is of type ‘EJB’.

```
<ds_ejb name="record_on_weblogic5">
<contextProperty name="cp1" value="cp1_value"/>
       <contextProperty name="cp2" value="cp2_value"/>
</ds_ejb>
```

– ‘name’ is the ejb data source's name
– ‘name’ and ‘value’ within the <contextProperty> tags are the context property’s name and value, respectively

<ds_xml>

This tag is to be used when the data source is of type ‘XML’.

```
<ds_xml name="xml1" url="xml_url" method="post"/>
```

– ‘name’ is the xml data source's name
– ‘url’ is the xml data source's new url
– ‘method’ is the xml data source's new method: ["get"|"post"|""]

<ds_soap>

This tag is to be used when the data source is of type ‘SOAP’.

```
<ds_soap url="http://localhost:8080/axis/services/
TempConvertWS"
serverclass="localhost.axis.services.TempConvertWS.TempConv
ertWSSoapBindingStub" requireLogin="true" user="Tom"
password="enter">
```

– ‘`url`’ is the soap data source's new url
– ‘`serverclass`’ is the soap data source's new server class
– ‘`requireLogin`’ is true if the soap data source requires login
– ‘`user`’ is the soap data source's new user if requires login
– ‘`password`’ is the soap data source's new password if requires login

APPENDIX E: Exploding Configuration Files

By default, reusable report components are stored in a zip file called *stylereport.srl*, and Viewsheets and Worksheets are stored in a binary file called *asset.dat*. If necessary, you can explode these bundle files into their individual components. The following sections explain how to do this.

E.1 Exploding asset.dat into Individual Components

By default, Viewsheets and Worksheets are stored in a binary file called *asset.dat*. You can explode this bundle file into its individual components through an Ant task or through the command line.

To explode the *asset.dat* file into individual components from the command line, follow the steps below:

1. Make sure that the Style Intelligence JAR files (e.g., *bisuite.jar* and *etools.jar*) are on your classpath. For example:

   ```
   set
   classpath=%CLASSPATH%;path_to\bisuite.jar;path_to\etools.ja
   r
   ```

2. Make sure that the Ant JAR file is on your classpath. (This is required even if you are using the command line.)

   ```
   set classpath=%CLASSPATH%;path_to\ant.jar
   ```

3. Enter the following command to extract the files:

   ```
   java inetsoft.sree.adm.ant.ExtractStorageTask path_to/
   asset.dat
   ```

This explodes the specified *asset.dat* file into a directory called *asset.dat.d* in the same location. Whenever this *asset.dat.d* directory is in existence, Style Studio and the Style Intelligence server read and write assets from the directory rather than from the binary *asset.dat* file.

To accomplish the same result using an Ant task, include the following code in the Ant script:

1. Specify the path to Style Intelligence JAR files:

   ```
   <typedef resource="inetsoft/sree/adm/ant/antlib.xml">
     <classpath>
       <include name="path_to/bisuite.jar"/>
       <include name="path_to/etools.jar"/>
     </classpath>
   </typedef>
   ```

2. Extract the files:

```
<extractstorage src="path_to/asset.dat"/>
```

To recreate the *asset.dat* file from *asset.dat.d* directory, substitute the following command line or Ant code into the corresponding instructions above.

```
// Command Line
java inetsoft.sree.adm.ant.PackageStorageTask path_to/
asset.dat.d

// Ant script
<packagestorage src="path_to/asset.dat.d"/>
```

E.2 Exploding stylereport.srl into Individual Components

By default, reusable report components are stored in a zip file called *stylereport.srl* in the SREE Home directory. You can explode this bundle file into its individual components through an Ant task or through the command line.

To explode the *stylereport.srl* file into individual components from the command line, follow the steps below:

1. Verify that the *stylereport.srl* file is located in the SREE Home directory (*WEB-INF/classes*, by default).

2. Make sure that the Style Intelligence JAR files (e.g., *bisuite.jar* and *etools.jar*) are on your classpath. For example:

```
set
classpath=%CLASSPATH%;path_to\bisuite.jar;path_to\etools.ja
r
```

3. Make sure that the Ant JAR file is on your classpath. (This is required even if you are using the command line.)

```
set classpath=%CLASSPATH%;path_to\ant.jar
```

4. Enter the following command to extract the files:

```
java inetsoft.sree.adm.ant.ExtractLibraryTask path_to/
stylereport.srl
```

This explodes the specified *stylereport.srl* file into a directory called *stylereport.srl.d* in the same location. Whenever this *stylereport.srl.d* directory is in existence, Style Studio and the Style Intelligence server read and write assets from the directory rather than from the *stylereport.srl* zip file.

To accomplish the same result using an Ant task, include the following code in the Ant script:

1. Verify that the *stylereport.srl* file is located in the SREE Home directory (*WEB-INF/classes*, by default).

2. Specify the path to Style Intelligence JAR files:

```
<typedef resource="inetsoft/sree/adm/ant/antlib.xml">
  <classpath>
    <include name="path_to/bisuite.jar"/>
    <include name="path_to/etools.jar"/>
  </classpath>
</typedef>
```

3. Extract the files:

```
<extractlibrary src="path_to/stylereport.srl"/>
```

To recreate the *stylereport.srl* file from *stylereport.srl.d* directory, substitute the following command line or Ant code into the corresponding instructions above.

```
// Command Line
java inetsoft.sree.adm.ant.PackageLibraryTask path_to/
stylereport.srl.d

// Ant script
<packagelibrary src="path_to/stylereport.srl.d"/>
```

APPENDIX F: Creating a Custom Map

If you have *shapefile* data that describes particular geographical regions, you can use this shapefile to create a custom map chart for those regions.

The following sections describe the general procedure for converting a shapefile into a map that can be used by the Style Intelligence server. This process uses the open-source application OpenJUMP, available from the OpenJUMP website. For complete information about the features of OpenJUMP, including drawing arbitrary shapes, please consult the OpenJUMP documentation.

F.1 Installing OpenJUMP and the InetSoft Extension

The open-source OpenJUMP GIS application allows you to convert data from a shapefile into a map resource that can be used by the Style Intelligence server. To install the OpenJUMP GIS application and the InetSoft OpenJUMP extension, follow the steps below:

1. Download the free OpenJUMP GIS application from the OpenJUMP website. This open-source application allows you to import and edit your shape data.

2. Download the InetSoft OpenJUMP extension (*inetsoft.zip*) from http://chart.inetsoft.com/map/inetsoft.zip.

3. Extract the contents of the *inetsoft.zip* file. The archive contains a *lib/ext* directory with the following JAR files (version numbers may differ).

   ```
   colt-1.2.0.jar
   concurrent-1.3.4.jar
   inetsoft.jar
   ```

4. Copy these files into the *lib/ext* directory in the OpenJUMP installation folder.

The OpenJUMP application and InetSoft extension are now ready to use.

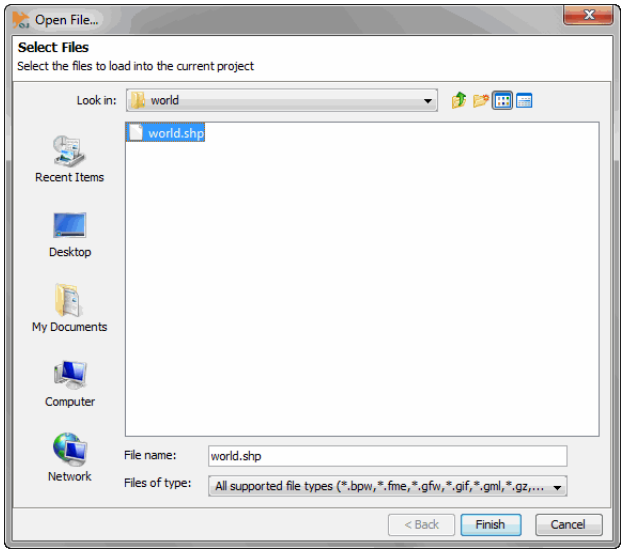
F.2 Exporting a Shape File as a Style Intelligence Map

To prepare and export a shape file for use with the Style Intelligence, follow the steps below.

1. Copy the shapefile (*.shp*) and any associated files (*.dbf*, *.shx*, etc.) to a location that is accessible to the OpenJUMP application.

2. Launch the OpenJUMP application.

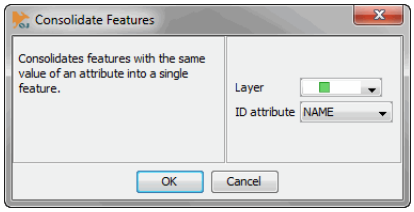
3. Choose ‘Open File’ from the OpenJUMP ‘File’ menu. Select the shapefile that you wish to export, and press ‘Finish’. (In this example, *world.shp* is a shapefile containing the political boundaries of all countries in the world.)



Consolidation merges all shapes for a atomic entity (e.g., a single country) into a single feature.

4. Consolidate the logical map features. To do this, follow the steps below:

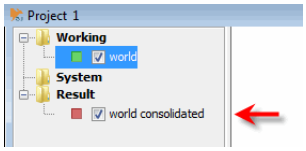
   a. Select ‘Consolidate Features’ from the ‘InetSoft’ menu. This opens the ‘Consolidate Features’ dialog box.



   b. In the ‘Layer’ menu, select the desired geographical layer.

   c. In ‘ID attribute’ menu, select the attribute that uniquely identifies the logical map features. (In this example, this is the country name.)

   d. Press ‘OK’. This creates the consolidated map, which is listed under the ‘Result’ node in the OpenJUMP project tree.



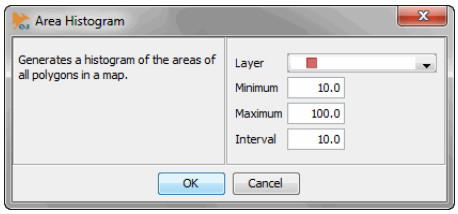
e. (Optional) To conserve memory, delete the original 'world' layer from the 'Working' node of the OpenJUMP project tree. To do this, right click on the 'world' entry, and choose 'Remove Selected Layers'.

6. (Optional) If you are creating a map of the United States, select 'Create U.S. inset' from the 'InetSoft' menu to inset the regions for Alaska and Hawaii. (This yields a transformed view of Alaska and Hawaii commonly used in published U.S. maps.)

7. To reduce the memory required by the map, remove polygons that have areas smaller than a specified size. To do this, follow the steps below:

   a. Select 'Area histogram' from the 'InetSoft' menu. This opens the 'Area Histogram' dialog box.

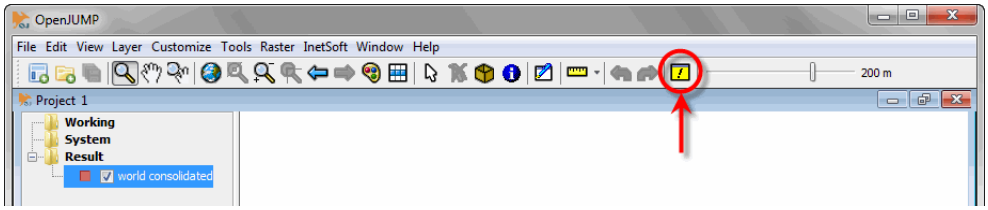
   b. Select the desired 'Layer', and set the 'Minimum', 'Maximum', and 'Interval' properties of the histogram.



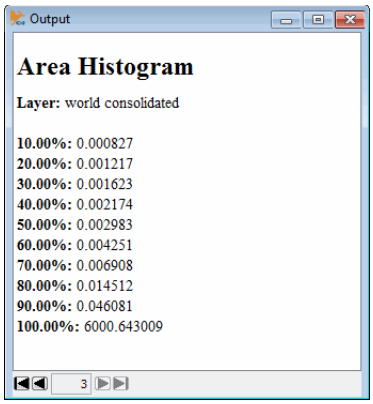
   For example, a minimum of 10%, maximum of 100%, and interval of 10% generates histogram values for the following percentages: 10, 20, 30, 40, 50, 60, 70, 80, 90, 100.

   c. Click 'OK' to generate the histogram.

   d. To view the histogram, press the 'Output Window' button in the OpenJUMP toolbar.

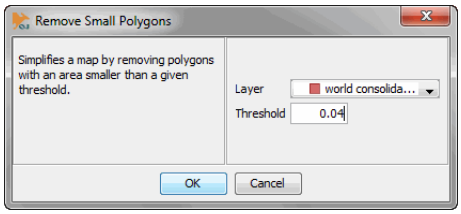


The histogram indicates the percentage of the polygons in the map which would be removed for a given area threshold. Use the histogram to identify the proper area threshold for removing polygons. For example, the histogram below indicates that by removing all polygons smaller than an area threshold of 0.046081, you could effectively remove 90% of polygons in the map.



In this example, areas in the range of 0.04 units are probably not visually significant compared to other areas, the largest of which is approximately 6000 units. Therefore, by using a polygon removal threshold of 0.04 you can eliminate the majority of polygons without detectably altering the displayed map.

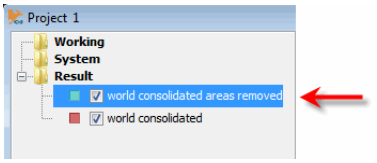
e. Select 'Remove small polygons' from the 'InetSoft' menu. This opens the 'Remove Small Polygons' dialog box.



Note: The most significant (largest) polygon of a map feature (e.g., country) is always retained regardless of the threshold.

f. Select the desired 'Layer', and enter a 'Threshold'. Polygons with areas smaller than the threshold will be removed from the map. In this case, a value of 0.04 will result in the removal of approximately 90% of the map's polygons.

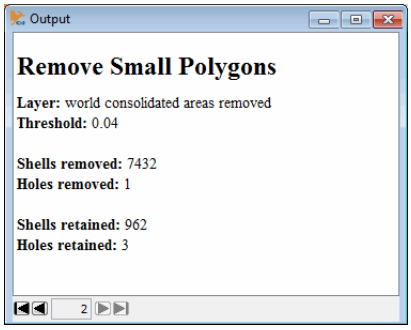
g. Press 'OK' to initiate the polygon removal. This creates a new map layer ("world consolidated areas removed" in the example).



h. (Optional) To remove the original layer ("world consolidated"), right-click the "world consolidated" node, and choose 'Remove Selected Layers' from the context menu.

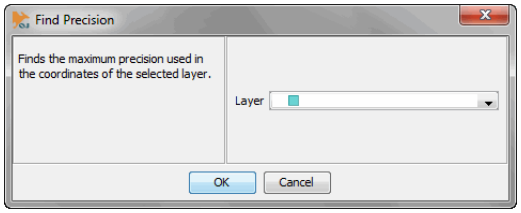
You may need to experiment with different area thresholds to determine which yields the best results.

i. To view the report for the polygon removal operation, press the 'Output Window' button in the OpenJUMP toolbar.



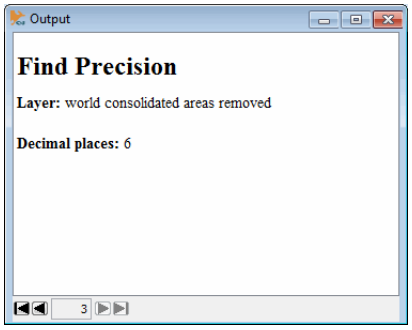
10. To help reduce the memory required for the map, determine the maximum precision to be used. Follow the steps below:

a. Select 'Find Precision' from the 'InetSoft' menu. This opens the 'Find Precision' dialog box.



b. Select the desired layer, and press 'OK'.

c. Press the 'Output Window' button in the OpenJUMP toolbar to view current precision. (The precision is expressed as the number of decimal places used to represent the map coordinates.)

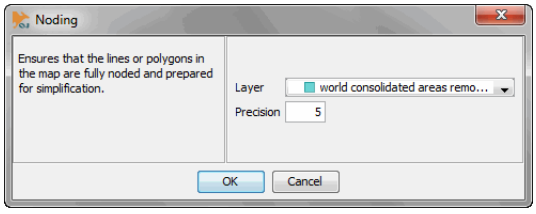


If the precision is unnecessarily high, you can reduce it in the next step. (A precision of six decimal places is more than adequate for most cases.)

4. Prepare the map for simplification. Follow the steps below:

   a. Select 'Node lines and polygons' from the 'InetSoft' menu. This opens the 'Noding' dialog box.

   b. Select the desired layer, and enter a value for 'Precision'. (A precision of six decimal places is more than adequate for most cases.)



   c. Press 'OK'. This snaps all coordinates to the specified precision and corrects some common topological errors.

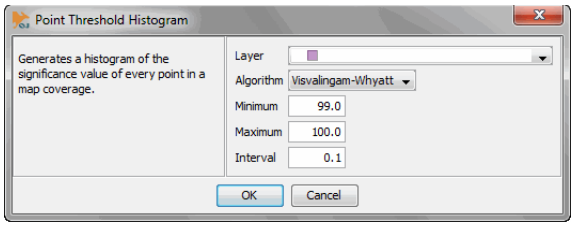
   d. (Optional) To remove the original layer ("world consolidated areas removed"), right-click the "world consolidated areas removed" node, and choose 'Remove Selected Layers' from the context menu.

5. Simplify lines and polygon boundaries by removing insignificant points. Follow the steps below:

   a. Select 'Point threshold histogram' from the 'InetSoft' menu. This opens the 'Point Threshold Histogram' dialog box.

   b. Select the desired 'Layer', and set the 'Minimum', 'Maximum', and 'Interval' properties of the threshold histogram.

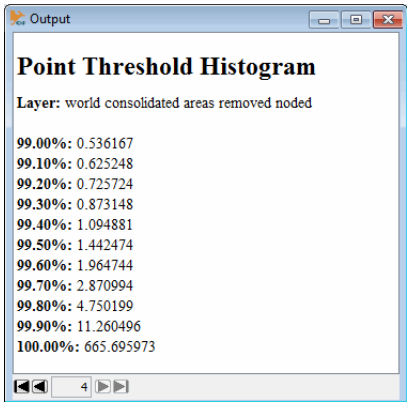
For example, a minimum of 99%, maximum of 100%, and interval of 0.1% generates histogram values for the following percentages: 99.0, 99.1, 99.2, 99.3, 99.4, 99.5, 99.6, 99.7, 99.8, 99.9, 100.



c. Select the desired simplification method in the 'Algorithm' menu. In most cases the 'Visvalingam-Whyatt' algorithm produces superior results. However, the 'Douglass-Peucker' algorithm may be faster.

d. Press 'OK'. This generates a histogram of boundary point significance.

e. To view the histogram, press the 'Output Window' button in the OpenJUMP toolbar.



The values in the histogram represent the point "significance," as determined by the simplification algorithm.

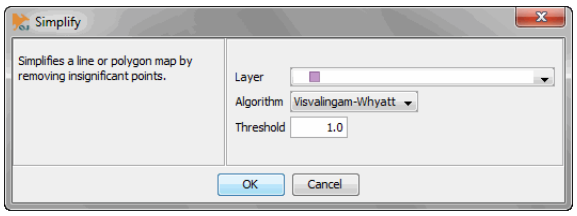
The histogram indicates the percentage of boundary points which would be removed for a given point "significance" threshold. Use the histogram to identify the proper point significance threshold for border simplification. For example, the histogram above indicates that by removing all points with significance smaller than 1.094881, you would effectively remove 99.4% of the boundary points.

In this example, points with significance in the range of 1.0 are probably not visually important compared to other points, the most significant of which have a value of approximately 665.

Therefore, by using a point removal threshold of 1.0 you could eliminate the majority of points without detectably altering the displayed map.

6. Run the simplification algorithm. Follow the steps below:

   a. Select ‘Simplify lines’ from the ‘InetSoft’ menu. This opens the ‘Simplify’ dialog box.



   b. Select the desired ‘Layer’, and choose the same ‘Algorithm’ that you used in the previous step.

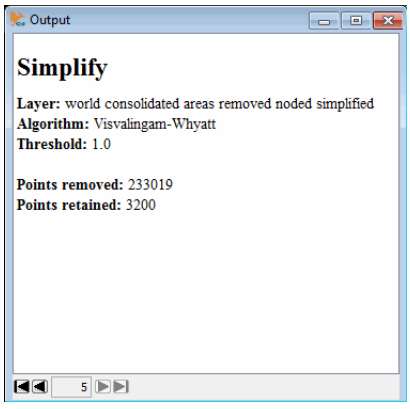
   c. Enter a ‘Threshold’ value. Boundary points with significance smaller than the threshold will be removed from the map. (In this case, a value of 1.0 will result in the removal of approximately 99% of the map’s points.)

   d. Press ‘OK’ to start the simplification.

   e. (Optional) To remove the original layer (“world consolidated areas removed noded”), right-click the “world consolidated areas removed noded” node, and choose ‘Remove Selected Layers’ from the context menu.

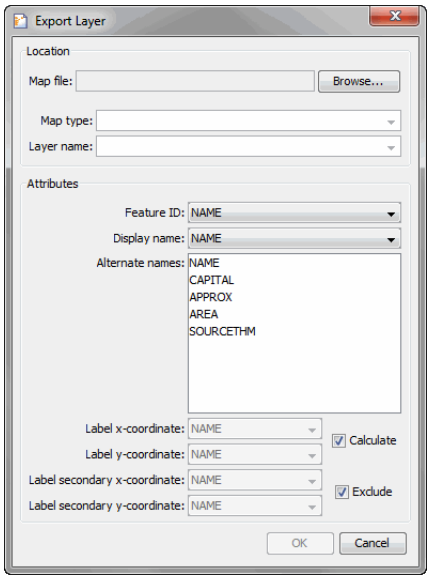
You may need to experiment with different point thresholds to determine which yields the best results.

   f. To view the report for the point removal operation, press the ‘Output Window’ button in the OpenJUMP toolbar.



7. Export the map to Style Intelligence. Follow the steps below:

a. Select the desired layer in the OpenJUMP project tree. Then select ‘Export layer’ from the ‘InetSoft’ menu. This opens the ‘Export Layer’ dialog box.



b. Specify the ‘Map file’ location. The map file must be named “`mapdata.xml`” and must be located in the server repository directory (by default, *sree/WEB_INF/classes*). If the file does not exist, browse to the repository directory and enter “`mapdata.xml`” in the ‘File Name’ field.

```
Do not specify a name for the ‘Map type’ that is identical to one
of the built-in map types: ‘U.S.’, ‘Asia’, ‘Canada’, ‘Europe’,
‘Mexico’, or ‘World’. Duplicating a built-in map type name will
cause an error.
```

c. Specify the ‘Map type’. This is the top-level category for the map. For example, a map of type “New Jersey” might have layers (specified below) such as “New Jersey Counties” and “New Jersey Zip Codes”.

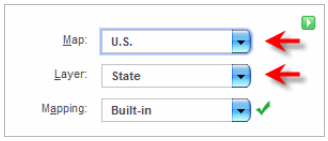
Select an existing layer to overwrite that layer.

d. Specify the ‘Layer name’. This is the name for the particular map you are exporting.

```
Do not specify a value for the ‘Layer name’ that is identical to
one of the built-in layers: ‘Country’, ‘Province’, ‘State’, ‘City’,
or ‘Zip’. Duplicating a built-in layer name will cause an error.
```

Note: The defined map type and any defined layers will appear in the menus of the Chart Editor’s geographical panel when you edit a geographical field in Style Intelligence. See *Binding a Chart to Geographical Data (Map)* in *Advanced Report Design*

and *Creating a Map Chart* in *Advanced Dashboard Design* for more details on using the Chart Editor.



e. Select the 'Feature ID'. This is the attribute that uniquely identifies the map features. (In some cases this may be the FIPS or ISO code for a feature.)

f. Select the 'Display name'. This is the attribute that contains the default, human-readable, names of the map features. (This may be the same as the Feature ID.)

g. (Optional) Select any desired 'Alternate names'. These are attributes that contain alternate display names for the map features. (These might be state abbreviations, ISO codes, etc.)

h. Enable the 'Calculate' option to automatically calculate the label coordinates as the centroids of the feature geometry. (If you do this, skip the following two steps). If you wish instead to use attribute values to control label positioning, disable this option.

i. If you have disabled the 'Calculate' option, specify for 'Label x-coordinate' the attribute that contains the x-coordinates (longitude) for feature label placement.

j. If you have disabled the 'Calculate' option, specify for 'Label y-coordinate' the attribute that contains the y-coordinates (latitude) for feature label placement.

k. Enable the 'Exclude' option if you do not want secondary label coordinates to be included in the exported data. (If you do this, skip the following two steps). If you wish to specify secondary coordinates, disable this option.

```
Note: Secondary coordinates are not currently used when rendering a
chart.
```

l. If you have disabled the 'Exclude' option, specify for 'Label secondary x-coordinate' the attribute that contains the secondary x-coordinates (longitude) for feature label placement.

m. If you have disabled the ‘Exclude’ option, specify for ‘Label secondary y-coordinate’ the attribute that contains the secondary y-coordinates (latitude) for feature label placement.

n. Press ‘OK’ to export the map data to the Style Intelligence repository directory.

15. Restart the Style Intelligence server. Your custom map will now be available in the ‘Map’ menu of the Chart Editor, along with the predefined maps.

F.3 Adding a Layer to a Custom Map

There are two ways to add multiple layers to a map. You can create multiple layers based on independent shape files (for example, one layer based on a U.S. *county* shape file, and another layer based on a U.S. *state* shape file). You can also use the OpenJUMP software to create arbitrary larger regions composed from the original regions of a single shape file. The following sections explain these approaches.

F.3.1 Adding a Layer Based on a New Shape File

To add a layer from a new shapefile to an existing map (for example, to add a “CensusRegions” layer to a “US2” map that already has a “state” layer), follow the steps below:

1. Obtain a shapefile that contains the data for the desired new layer.

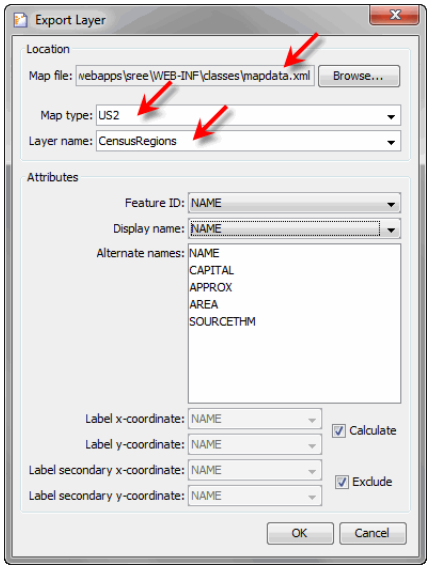
2. Proceed through the steps described in *Exporting a Shape File as a Style Intelligence Map* to process the shapefile appropriately.

3. When you select ‘Export layer’ from the ‘InetSoft’ menu, make the following settings:

   a. For the ‘Map file’, select the same *mapdata.xml* file that already contains the other layer(s) of the map.

   b. For the ‘Map type’, use the menu to select the map to which you want to add the layer (e.g., ‘US2’).

   c. For the ‘Layer name’, enter the name of the desired new layer (e.g., ‘CensusRegions’).



d. Complete the additional steps described in *Exporting a Shape File as a Style Intelligence Map*, and press 'OK'.

This exports the new layer into the existing *mapdata.xml* file, and associates the layer with the existing map.

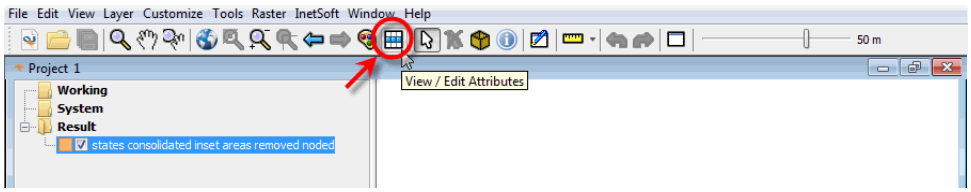
F.3.2 Adding a Layer Based on Merged Regions

To add a new layer by merging together regions within an existing shapefile (for example, by merging together states in a "US2" map to form "CensusDivisions"), follow the steps below:

1. Open the shapefile containing the regions that you want to merge in OpenJUMP.

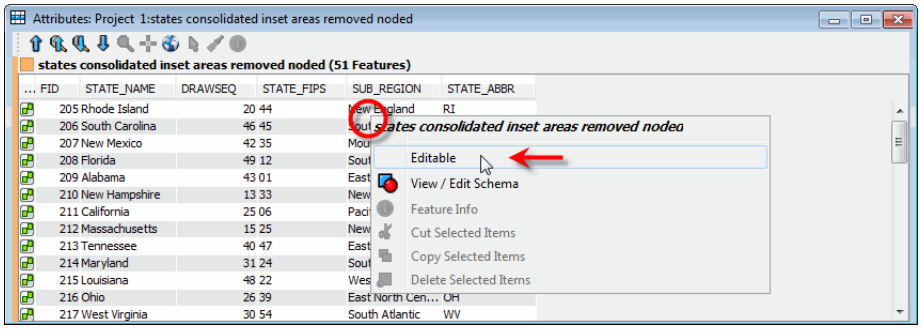
2. Perform any necessary simplification processing as described in *Exporting a Shape File as a Style Intelligence Map*.

3. Press the 'View/Edit Attributes' button in the OpenJUMP toolbar.



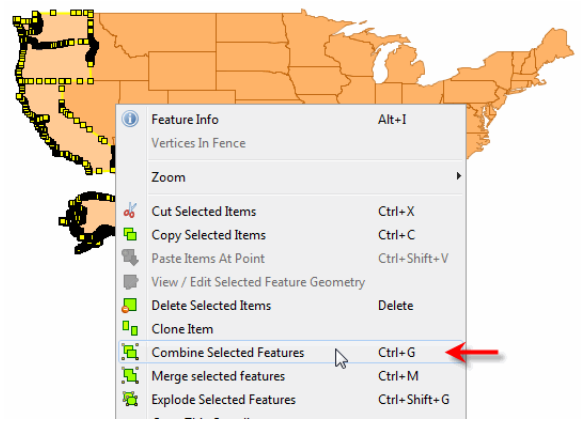
This displays the 'Attributes' table for the map.

4. Right-click on the table and select 'Editable' from the context menu.



5. Return to the map view. Shift-click to select the regions that you want to merge. For example, Shift-click to select all states belonging to the "Pacific Division" (California, Oregon, Washington, Alaska, Hawaii).

6. Right-click on one of the selected states, and choose 'Combine Selected Features'.



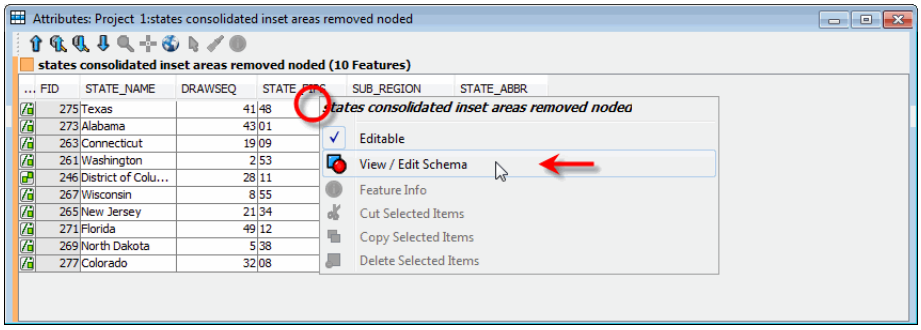
The selected states are now considered a single unit.

7. Repeat the above steps to create any additional desired units. (In this example, we combined states to create the additional census divisions: New England, Middle Atlantic, East North Central, West North Central, South Atlantic, East South Central, West South Central, and Mountain.)

8. (Optional) If necessary, create a new column to label the combined features. Follow the steps below:

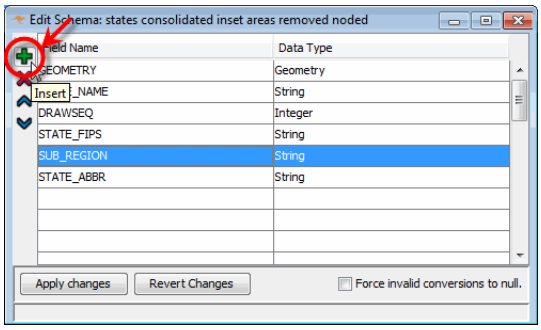
   a. Press the 'View/Edit Attributes' button in the OpenJUMP toolbar to display the 'Attributes' table.

   b. Right-click on the table, and select 'View/Edit Schema'.

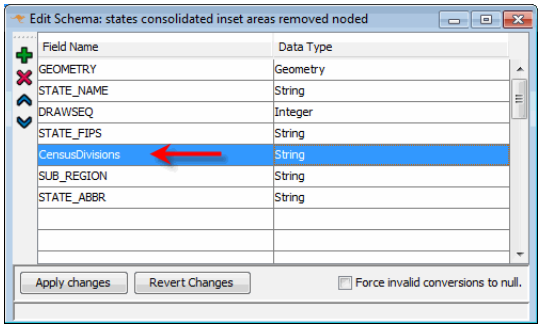


This opens the 'Edit Schema' dialog box.

c. Click in the table, and then press the 'Insert' (+) button to create a new column.

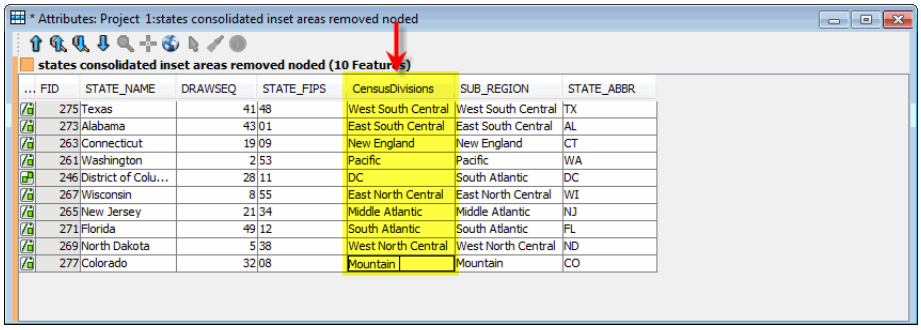


d. Enter the name and type of the new column (e.g., "CensusDivisions").



e. Press 'Apply changes' and return to the 'Attributes' table. Note that a new column has been added.

f. Enter the appropriate labels into the blank cells.



... FID	STATE_NAME	DRAWSEQ	STATE_FIPS	CensusDivisions	SUB_REGION	STATE_ABBR
275	Texas	41	48	West South Central	West South Central	TX
273	Alabama	43	01	East South Central	East South Central	AL
263	Connecticut	19	09	New England	New England	CT
261	Washington	2	53	Pacific	Pacific	WA
246	District of Colu...	28	11	DC	South Atlantic	DC
267	Wisconsin	8	55	East North Central	East North Central	WI
265	New Jersey	21	34	Middle Atlantic	Middle Atlantic	NJ
271	Florida	49	12	South Atlantic	South Atlantic	FL
269	North Dakota	5	38	West North Central	West North Central	ND
277	Colorado	32	08	Mountain	Mountain	CO

Note that when you click on a blank cell, the corresponding map region is highlighted.

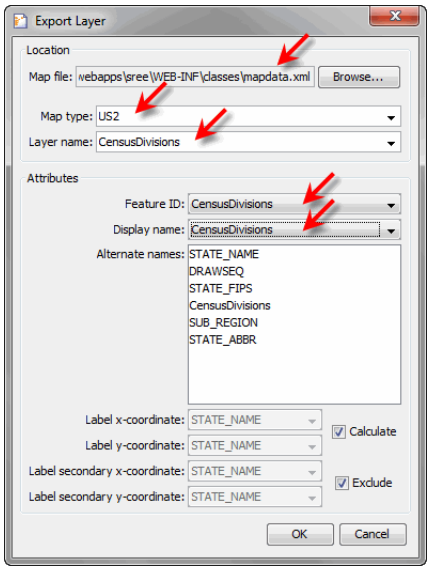
9. With the new column defined, proceed with the instructions in *Exporting a Shape File as a Style Intelligence Map*. When you select 'Export layer' from the 'InetSoft' menu, make the following settings:

   a. For the 'Map file', select the same *mapdata.xml* file that already contains the other layer(s) of the map.

   b. For the 'Map type', use the menu to select the map to which you want to add the layer (e.g., 'US2').

   c. For the 'Layer name', enter the name of the desired new layer (e.g., 'CensusDivisions').

   d. For the 'Feature ID' and 'Display name', select the new column that you added to the 'Attributes' table.



e. Complete the additional steps described in *Exporting a Shape File as a Style Intelligence Map*, and press 'OK'.

This exports the new layer into the existing *mapdata.xml* file, and associates the layer with the existing map.

F.4 Overlaying a Shape on a Custom Map

In some cases you may want to overlay shapes on top of an existing custom map. For example, you may want to overlay shapes corresponding to the four Census Regions on top of an existing U.S. map that already displays data using a 'State' layer. In this case, **do not** add the Census Regions as a new layer to the existing map as described in *Adding a Layer to a Custom Map*, because a given map can only have a single data layer. (You would then be forced to choose *either* 'State' or 'CensusRegions' as the displayed layer.)

To create a map that displays data on the 'State' layer, but that also draws the Census Regions (without data), follow the steps below:

1. In OpenJUMP, open the shapefile containing the shapes that you want to overlay.

   To manually create shapes by merging existing shapes (e.g., to create larger regions by merging multiple states) follow the steps described in *Adding a Layer to a Custom Map*, but use the 'Merge selected features' option rather than 'Combine selected features'. **Do not** export the resulting map.

2. (Optional) Perform any necessary simplification processing as described in *Exporting a Shape File as a Style Intelligence Map*.

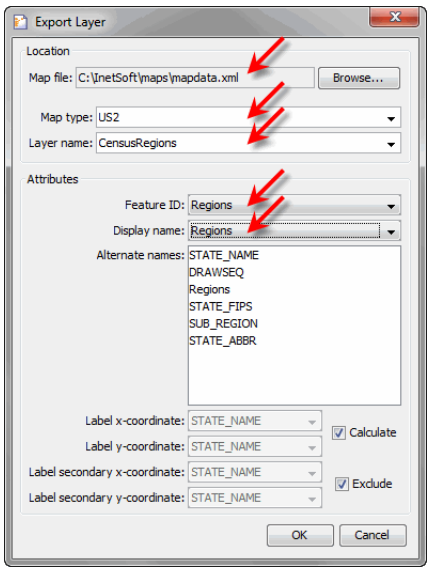
3. Select 'Export layer' from the 'InetSoft' menu, make the following settings:

   a. For the 'Map file', press the 'Browse' button. Select a **different location** than your current SREE Home (*WEB-INF/classes*) so that you do not overwrite the existing *mapdata.xml* file. For example, create a new "maps" folder into which to save the export.

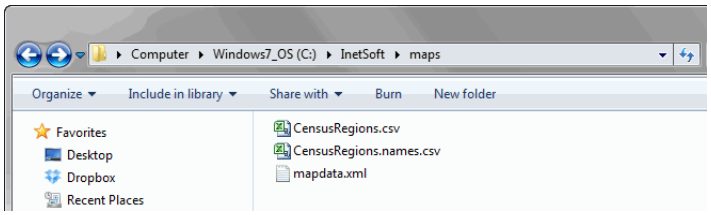
   b. For the 'Map type', enter a map name (e.g., 'US2').

   c. For the 'Layer name', enter a name for the layer you are exporting (e.g., 'CensusRegions').

d. For the 'Feature ID' and 'Display name', select the appropriate feature column.



e. Press 'OK'. This exports the layer to the location that you specified. The saved files include two CSV files and a *mapdata.xml* file.



4. Copy the two CSV files from the export location to your SREE Home directory (*WEB-INF/classes*). **Do not** copy the *mapdata.xml* file.

5. Stop the Style Intelligence server (if it is running).

6. Open the original *mapdata.xml* file in the SREE Home directory with a text editor. Amend *mapdata.xml* to allow the custom map to access the saved CSV files, and provide the appropriate styles for the labels and outlines. A example follows:

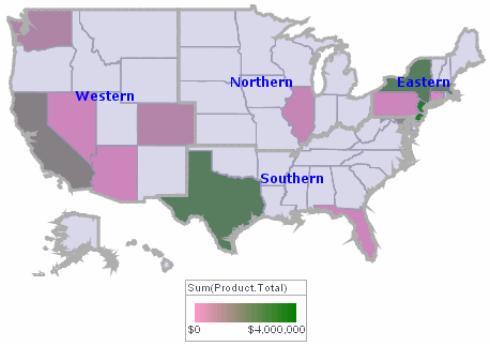
```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<mapdata>
  <style name="label">
    <font font="Verdana-BOLD-12" color="#0000FF"/>
    <line width="3" dash="1"/>
    <fill color="#0000FF" alpha=".1"/>
  </style>
```

```
  <nametable name="US_States_names" file="/
US_States.names.csv" labels="0"/>
  <layer name="US_States" id="104"
nametable="US_States_names"/>
  <matching layers="104" columns="0"/>

  <map type="US2">
    <shapedata data="/US_States.csv" layer="US_States">
      <shapes style="label" data="/CensusRegions.csv"/>
      <labels data="/CensusRegions.names.csv"/>
      <sampledata>Rhode Island</sampledata>
      <sampledata>South Carolina</sampledata>
    </shapedata>
  </map>
</mapdata>
```

7. Save the *mapdata.xml* file, and restart the Style Intelligence Server.

8. Create a map chart in Style Studio or Visual Composer based on the custom map. Observe that the specified regions and labels are overlaid on the map as desired.



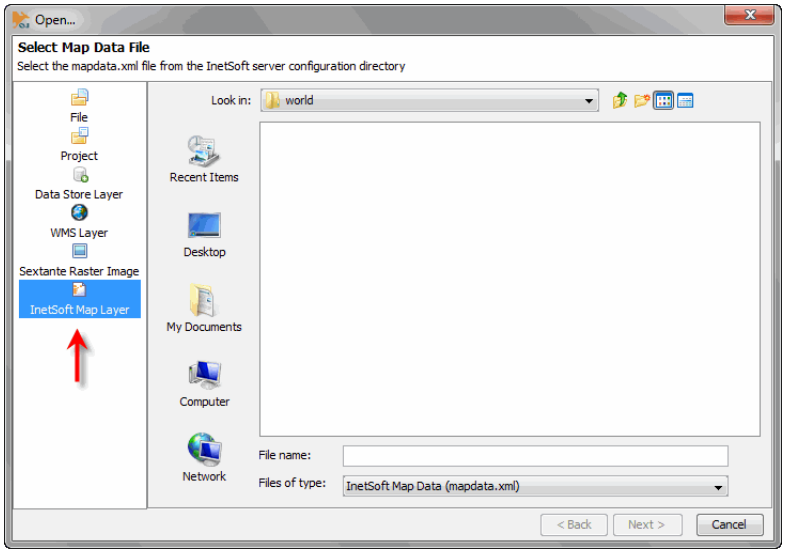
F.5 Editing a Custom Map

To edit map data from Style Intelligence in the OpenJUMP application, follow the steps below:

1. Install the OpenJUMP application and InetSoft extension as described in *Installing OpenJUMP and the InetSoft Extension*.

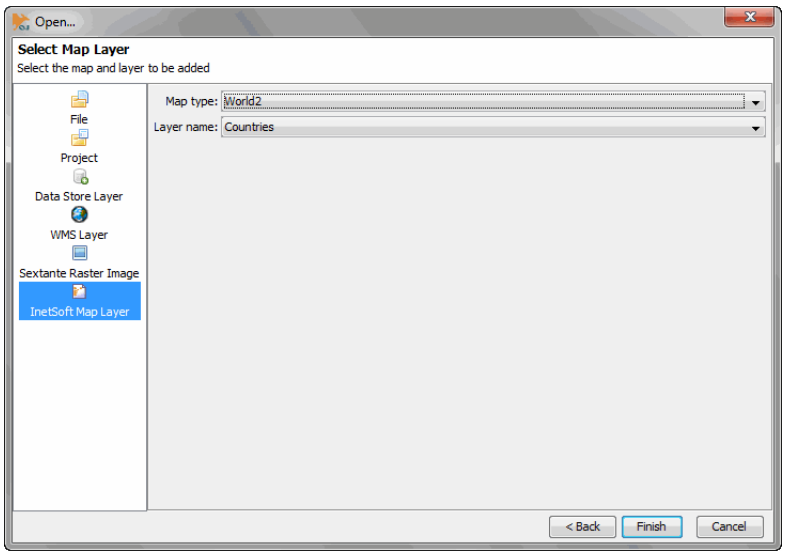
2. Launch OpenJUMP, and select ‘Open’ from the ‘File’ menu.

3. Select ‘InetSoft Map Layer’ in the left panel.



4. Navigate to the server repository directory (by default, sree/WEB-INF/classes), and select the *mapdata.xml* file. Press 'Next'.

5. Select the 'Map type' and 'Layer name' that you want to edit. (See *Exporting a Shape File as a Style Intelligence Map* for information about the map type and layer name.)



6. Press 'Finish'. This loads the map data into the OpenJUMP application.

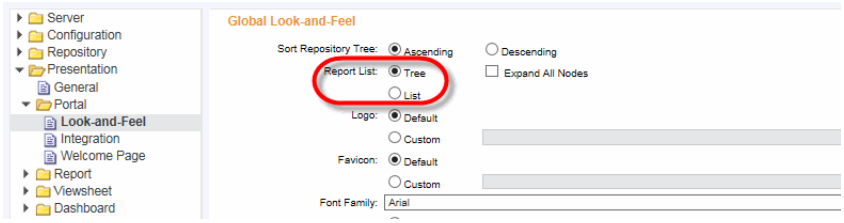
7. Edit the map as desired, and export the modified map. Follow the procedure described in *Exporting a Shape File as a Style Intelligence Map*.

APPENDIX G: Section 508 Accessibility Compliance

Note: Enterprise Manager is not Section 508-compliant.

The User Portal is compliant with Section 508 of the federal Rehabilitation Act of 1973. This means that most features of the User Portal are accessible via keyboard commands and corresponding voice commands. To enable full Section 508 compliance in the User Portal, set `accessibility.enabled=true` in *sree.properties*.

Note that the User Portal 'Repository' listing is fully accessible only in 'List' view, not in 'Tree' view. Configure the Repository for 'List' view under the *Look-and-Feel* page of Enterprise Manager.



See *Designing Reports for Section 508 Compliance* in *Advanced Report Design* for accessibility factors that report designers should consider.

See Also

Keyboard Navigation, for information on dashboard accessibility.

Advanced Server Integration

One of the greatest advantages of Style Intelligence is its flexibility and adaptability; it does not limit you to a single deployment or presentation framework. Instead, you can seamlessly integrate Style Intelligence components into your unique application at many different levels.

Because Style Intelligence is implemented entirely in Java, it is immediately compatible with any J2EE compliant environment. You can make use of web services, customize the built-in portal, and pass credentials directly from your application to the Style Intelligence framework for single sign-on.

Our reliance on open standards makes it possible to integrate with non-Java environments as well. It is possible to retrieve reports from simple URL requests, and Web Services.

This document discusses these and other ways to integrate Style Intelligence into an existing environment.

1 Integrating Style Intelligence into Your Web Application

There are three primary tasks required to integrate Style Intelligence into a 3rd party-web application:

1. *Integrating the Web User Interface*
2. *Configuring Single Sign-On (SSO)*
3. *Selecting and Configuring a Security Provider*

The following sections guide you through this process.

1.1 Integrating the Web User Interface

The easiest way to embed a Style Intelligence module into your web application is to use an *IFrame*. This approach allows you to selectively embed portions of the Style Intelligence User Portal, as well as individual Reports and Viewsheets, within the HTML, JSP, or ASP pages of your application. The following sections explain the available IFrame embedding options.

An alternative to IFrame embedding is JSP-based integration (see Appendix D: *Configuring Style Intelligence for JSP Tags*), but there are several advantages that make IFrames the preferred approach in most cases:

- IFrames are easier to use, and Style Intelligence's report design architecture is geared toward IFrame integration. JSPs can be time-consuming to build and require knowledge of the JSP Tag Library.
- Reports that are integrated using JSP technology do not offer the end user Ad Hoc reporting functionality or portal utilities. These features are fully retained in IFrame-integrated reports.
- Certain application servers (WebLogic) require specific configurations and settings when using the JSP Tag Library. This complexity is avoided in the IFrame approach.

See Also
Appendix C: *Servlet Repository Parameters*, for useful "op" codes.

1.1.1 Embedding the Entire Style Intelligence Portal

To embed the entirety of the Style Intelligence User Portal within an IFrame, use the following syntax:

Replace 'remote-host:8080' with an actual machine name and port.

```
<iframe src="http://remotehost:8080/sree/Reports">
<p>Text for obsolete browsers.</p>
</iframe>
```

1.1.2 Embedding a Tab of the Style Intelligence Portal

To embed a single tab of the Style Intelligence User Portal within an IFrame, use the following syntax,

Replace 'remote-host:8080' with an actual machine name and port.

```
<iframe src="http://remotehost:8080/sree/
Reports?op=portal_{TabName}">
<p>Text for obsolete browsers.</p>
</iframe>
```

where *TabName* is the title of the desired tab. For example, to embed the contents of the **Report** tab of the Portal, configure the IFrame as follows:

```
<iframe src="http://remotehost:8080/sree/
Reports?op=portal_report">
<p>Text for obsolete browsers.</p>
</iframe>
```

This embeds both the left-side Repository Tree and the right-side report display area.

See Also
Creating a Custom Report List, for advanced API-based integration.

1.1.3 Embedding a Report or Viewsheet

To embed an individual report into an IFrame, use the following syntax,

Replace 'remote-host:8080' with an actual machine name and port.

```
<iframe src="http://remotehost:8080/sree/
Reports?op=frameReplet&name=FolderName/
ReportName&outtype=REPORT">
<p>Text for obsolete browsers.</p>
</iframe>
```

Use “%20” to represent a space in a report or Viewsheet name.

where `ReportName` is the name of the desired report, and `FolderName` is the name of the report’s containing folder in the Repository.

Similarly, to embed an individual Viewsheet into an IFrame, use the following syntax:

```
<iframe src="http://remotehost:8080/sree/
Reports?op=vs&path=/FolderName/ViewsheetName">
<p>Text for obsolete browsers.</p>
</iframe>
```

```
Note: The path does not include explicit reference to 'My Reports',
and does not begin with a '/'.
```

For a user-scope Viewsheet (i.e., a Viewsheet in the ‘My Reports’ folder), the `path` should begin with the username. For example, for user ‘`guest`’ and a Viewsheet called ‘`MyViewsheet`’ residing in ‘My Folder’ (within ‘My Reports’) the path should be specified as follows:

```
<iframe src="http://remotehost:8080/sree/
Reports?op=vs&path=guest/My%20Folder/MyViewsheet">
<p>Text for obsolete browsers.</p>
</iframe>
```

To embed the Viewsheet in an IFrame of an HTA (HTML Application) page, add parameter "`isHTA=true`". For example:

```
<iframe src="http://remotehost:8080/sree/
Reports?op=vs&path=/FolderName/ViewsheetName&isHTA=true">
<p>Text for obsolete browsers.</p>
</iframe>
```

To load the Viewsheet in Visual Composer, see *Embedding the Visual Composer*.

Example: Embedding a Report

This section provides a fuller example of IFrame embedding. the HTML markup below generates a simple web page with two divs, the top div containing a heading and some text, and the bottom div containing an IFrame with embedded report. To run this example, replace "`remotehost`" with the name of the remote server on which the report engine is running.

```
<html>
<head>
   <title>IFrame Integration Example</title>
   <meta http-equiv="content-type" content="text/html;
      charset=utf-8" />
<link href="hostStyle.css" rel="stylesheet" type="text/css" />
</head>

<body>
   <div id="TitleDiv">
   <h1>IFrame Integration Example</h1>
   <p>This is an example of IFrame integration.</p>
   </div>

   <div id="IFrameDiv">
   <iframe src="http://remotehost:8080/sree/
Reports?op=frameReplet&name=OrderList" frameborder="1" width="100%"
height="100%">
   <p>Text for obsolete browsers.</p>
   </iframe>
   </div>
</body>
</html>
```

Note: When the div surrounding an IFrame has its 'overflow' property set to 'auto', Internet Explorer may not correctly position the embedded report's context menus.

To control the style of the above HTML, including the size and placement of the div containing the IFrame, you can use a CSS file such as the one below (**`hostStyle.css`**).

```
html, body {
   margin:0;
   padding:0;
   height:100%;
   font-family: arial;
}
div {
   margin-left: auto;
   margin-right: auto;
   width: 75%;
   padding:0;
```

```
   background: white;
}
div#IFrameDiv {
   height: 75%;
   bottom:0;
}
div#TitleDiv {
   height: 20%;
   text-align:center;
}
```

¢

1.1.4 Embedding the Visual Composer

To embed the Visual Composer in an IFrame, use the following syntax:

```
<iframe src="http://remotehost:8080/sree/
Reports?op=vs&edit=true">
<p>Text for obsolete browsers.</p>
</iframe>
```

To embed the Visual Composer and open a particular Viewsheet for editing, use the following syntax:

```
<iframe src="http://remotehost:8080/sree/
Reports?op=vs&path=/FolderName/ViewsheetName&edit=true">
<p>Text for obsolete browsers.</p>
</iframe>
```

See Appendix C: *Servlet Repository Parameters* for other useful “op” codes.

1.2 Configuring Single Sign-On (SSO)

Single Sign-On (SSO) is the process of transparently passing user credentials from one application to another so that the user is only required to present login credentials a single time. For instance, consider the case where a user logs into their main company portal and clicks on a link called “reports” to access the User Portal. If single sign-on is in effect, the user can be automatically admitted to the Style Intelligence Portal without any additional authentication challenges. This helps to create a seamless user experience.

You can implement single sign-on capability in two ways:

- *Form-Based Single Sign-On*
- *Session-Based Single Sign-On*

The implementation that you choose depends on the nature of your web application.

1.2.1 Form-Based Single Sign-On

The form-based SSO method works for both J2EE and .NET applications. Your primary sign-on web application should generate a form containing hidden fields "`userid`" and "`password`", similar to the one below:

```
<form method=post name=reportForm action="/sree/Reports">
   <a onClick=reportForm.submit();>View Reports</a>
   <input type=hidden name=userid value=xxxx>
   <input type=hidden name=password value=yyyy>
</form>
```

When a user clicks on the link in your web application, the form transparently submits the parameters "`userid`" and "`password`" to the Style Intelligence server. These parameters are then passed to the Security Provider for authentication. Once the Security Provider authenticates the user's credentials, it assigns the user to the appropriate groups and roles, and furnishes the user with the authorized permissions. To add information to the SRPrincipal object, see *Accessing SRPrincipal via Login Listener*.

Next Steps: *Selecting and Configuring a Security Provider*.

1.2.2 Session-Based Single Sign-On

You can implement single sign-on by placing a Principal object in the user's HTTP session. This allows Style Intelligence to automatically sign in the user without attempting to authenticate their credentials.

The Style Intelligence *SRPrincipal* object is used to hold the security information, and you must add this object to the user's session as an attribute. The amount of information that you provide within the SRPrincipal object determines the extent of SSO, as well as the role and responsibility of the security provider.

You can implement SRPrincipal-based SSO for both .NET (non-Java) and J2EE applications:

- For J2EE applications, use the lightweight InetSoft Proxy Servlet within your own application. The proxy duplicates the SRPrincipal object onto the Style Intelligence web application. The recommended injection point for the SRPrincipal object is a request filter mapped to the Proxy Servlet. See *Request Filter mapped to the Style Intelligence Proxy Servlet*.
- For .NET (non-Java) applications, you need to pass the user information to the Style Intelligence web application via a suitable method such as cookies, request parameters, etc. The recommended injection point for the SRPrincipal object is a request filter mapped to the Style Intelligence Report Servlet. See *Request Filter mapped to the Style Intelligence Report Servlet*.

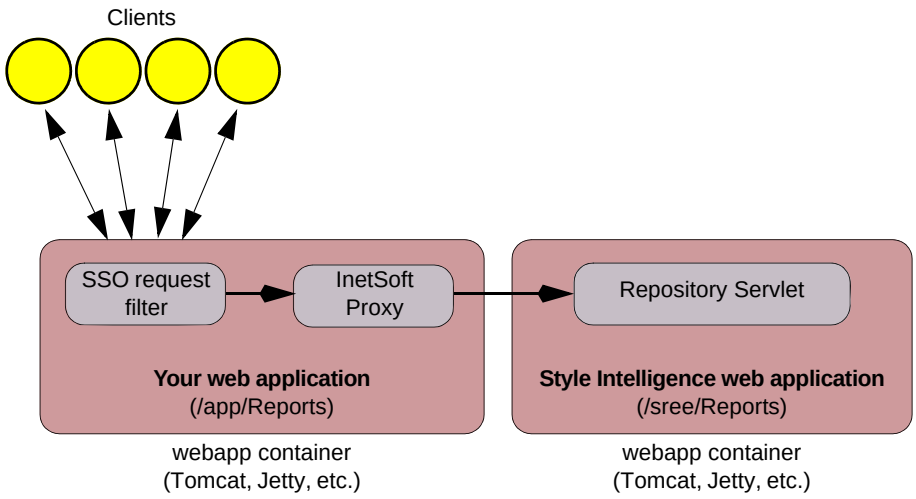
Request Filter mapped to the Style Intelligence Proxy Servlet

`The proxy servlet mapping procedure can only be used for J2EE apps.`

This proxy also serves as load balancer for clustered reporting environments. See *Configuring Server Clustering* in *Advanced Server Administration* for more details.

Style Intelligence provides a lightweight Proxy Servlet that can run within your own web application context. By adding an SRPrincipal object to the user session, you can pass user information to the Style Intelligence application. When a user performs a session-based sign-on to the Proxy, this will then effectively sign the user into the remote Style Intelligence server.

Figure 1. Typical architecture for SRPrincipal-based SSO with Proxy



This section explains how to deploy and configure the Proxy Servlet and the request filter so that session attributes are correctly mapped to the Proxy Servlet. Follow the steps below:

Note: You can rename *etools.jar* to *ztools.jar* to take advantage of the alphabetical loading of JARs in order to give preference to your application when loading common class libraries.

1. Verify that you have added the Style Intelligence product JARs *bisuite.jar* or *sree.jar* or *visual.jar*, and *etools.jar* to your web application within the following directory: *\webapps\{YourWebApp}\WEB-INF\lib*.

2. Deploy the Proxy Servlet in the deployment descriptor (*web.xml*) file of your web application by adding the following code:

```
<web-app>

  <servlet>
    <servlet-name>replets</servlet-name>
    <servlet-class>inetsoft.sree.web.ClusterServlet</
servlet-class>
  </servlet>

  <context-param>
    <param-name>sree.home</param-name>
```

```
    <param-value>{Any Local Path on Your App Server}</param-
value>
  </context-param>

  <servlet-mapping>
    <servlet-name>replets</servlet-name>
    <url-pattern>/Reports</url-pattern>
  </servlet-mapping>

  <servlet-mapping>
    <servlet-name>replets</servlet-name>
    <url-pattern>/Reports/dashboard/*</url-pattern>
  </servlet-mapping>

</web-app>
```

3. Configure the report server for cluster mode. See *Configuring Server Clustering* in *Advanced Server Administration*.

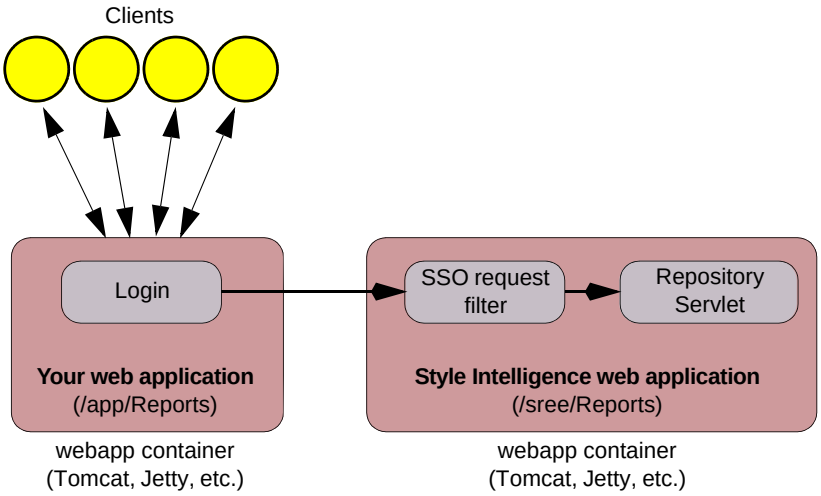
Next Steps: Create a request filter with mapping to the Proxy Servlet. See *Request Filter Example*.

Request Filter mapped to the Style Intelligence Report Servlet

`The report servlet mapping can be used for both J2EE and .NET apps.`

In cases where you cannot use the Proxy Servlet, such as non-J2EE applications (e.g., SharePoint portal or any .NET application uses an IIS server), you can still inject session attributes by using a request filter. However, this filter must run within the Style Intelligence web application.

Figure 2. Typical architecture for SRPrincipal-based SSO without Proxy



When a user clicks on a link (in your web application) that accesses the Style Intelligence Report Server, your application must pass the user information to the request filter. It can do this by using the HTTP request header or request parameters, or by using security frameworks such as

Spring Security or Oracle Identity Manager, etc. The request filter intercepts the request to the Style Intelligence Report Servlet, extracts the user information, injects the necessary SRPrincipal object, and performs the single sign-on.

Next Steps: Create a request filter with mapping to the Proxy Servlet. See *Request Filter Example*.

Request Filter Example

A *request filter* is a server module that intercepts all requests made to a servlet. Below is a complete example of a request filter that performs a single sign-on into the Style Intelligence Report Servlet or its Proxy Servlet. To use the request filter, compile the class below and place it on the classpath.

Please note that the credential-passing mechanism in this example is intended for demonstration only, and should not be used in a production environment. A ticketing mechanism is typically used for this purpose, and the request filter should include logic to validate the ticket.

```
package com.inetsoft.demo;

import java.io.IOException;
import javax.servlet.*;
import javax.servlet.http.*;
import inetsoft.sree.RepletRepository;
import inetsoft.sree.security.SRPrincipal;

public final class InetSoftSSOFilter implements Filter {
  private FilterConfig filterConfig = null;

  public void init(FilterConfig filterConfig)
  throws ServletException {
    this.filterConfig = filterConfig;
  }

  public void destroy() {
      this.filterConfig = null;
  }

  public void doFilter(ServletRequest request,
ServletResponse response, FilterChain chain)
  throws IOException, ServletException {
    if(request instanceof HttpServletRequest) {
      HttpServletRequest hrequest = (HttpServletRequest)
request;
      HttpSession session = hrequest.getSession();
      SRPrincipal prin =  (SRPrincipal)session.getAttribute
   (RepletRepository.PRINCIPAL_COOKIE);

      if (prin == null) {
```

This simplistic credential-passing mechanism is intended for demonstration only, and is **not** recommended for use in a production environment.

```
        // Extract user info based on your environment.
```

```
            // For demo purpose, we assume here that it is
            // passed as a URL parameter.

            String user = request.getParameter("SSO");

            if(user != null) {
              prin = new SRPrincipal(user);
              session.setAttribute(RepletRepository.PRINCIPAL_
COOKIE, prin);

            }
          }
        }

        chain.doFilter(request, response);
      }
    }
```

This example assumes that user information is passed via a request parameter called "SSO". The code in **bold text** performs a partial SSO, logging in the user without group or role assignment. The distinction between partial SSO and complete SSO is explained below.

Partial SSO: Logging In the User without a Group or Role Assignment

Partial SSO logs a user into the Style Intelligence web application without any group or role assignment. Style Intelligence will invoke the `getUser()` method in the authentication module of the security provider, which is responsible for looking up the user's group and role assignment.

To implement partial SSO, use the code snippet in **bold text** in the example `InetSoftSSOFilter` above.

```
prin = new SRPrincipal(user);
session.setAttribute(RepletRepository.PRINCIPAL_COOKIE,
prin);
```

Complete SSO - Logging In the User with a Group and Role Assignment

Complete SSO logs a user into the Style Intelligence web application with a specific group and role assignment. When you provide such group and/or role information within the SRPrincipal object, Style Intelligence uses the specified assignments and bypasses the authentication module of the security provider. In this case, the security provider is responsible only for providing a list of groups and roles to set security permissions.

To implement complete SSO, use the following code snippet in place of the **bold text** in the example `InetSoftSSOFilter` above.

```
prin = new SRPrincipal(user, new String[] {"role1",
"role2"}, new String[]{"Group1"}, 1234);
session.setAttribute(RepletRepository.PRINCIPAL_COOKIE,
prin);
```

Next Steps: *Deploying the Request Filter*.

Deploying the Request Filter

If you are using the Proxy Servlet, deploy the request filter (see *Request Filter Example*) in the deployment descriptor file (`web.xml`) of your own web application. Otherwise deploy it in the deployment descriptor file (`web.xml`) of the Style Intelligence web application.

Add the following code to the appropriate `web.xml` file following the initial `<web-app>` tag:

```
<filter>
  <filter-name>SSOFilter</filter-name>
  <filter-class>com.inetsoft.demo.InetSoftSSOFilter</
filter-class>
</filter>

<filter-mapping>
  <filter-name>SSOFilter</filter-name>
  <url-pattern>/Reports</url-pattern>
  <url-pattern>/Reports/dashboard/*</url-pattern>
</filter-mapping>
```

Next Steps: *Selecting and Configuring a Security Provider*.

1.3 Selecting and Configuring a Security Provider

The security provider is responsible for authentication and authorization. Choose your security provider based on your SSO implementation and your security framework.

See Also

Security, in *Advanced Server Administration*, for information on configuring security in Enterprise Manager.

1.3.1 Security Provider Functions

The security provider includes the following key functions for authentication and authorization:

Authentication:

- `authenticate()`: Validates user login.
- `getUser()`: Maps user to groups and roles.
- `getGroups()`, `getRoles()`: Provide list of users, roles, and groups.

Authorization:

- `getPermission()`, `setPermission()`: Gets and sets Style Intelligence-specific permissions for users, groups, or roles.

The SSO method that you use will determine which functions of the security provider are bypassed. The list below lists the conditions under which certain functions are invoked or bypassed:

Form Based SSO

Authentication:

- `authenticate()`: Invoked
- `getUser()`: Invoked
- `getGroups()`, `getRoles()`: Invoked

Authorization:

- `setPermission()`, `getPermission()`: Invoked

Session Based SSO

Authentication:

- `authenticate()`: Bypassed
- `getUser()`: Bypassed, depending on information in SRPrincipal object.
- `getGroups()`, `getRoles()`: Invoked, when you set permissions via the Enterprise Manager.

Authorization:

- `getPermission()`, `setPermission()`: Invoked

1.3.2 Selecting a Security Provider

The following sections provide guidance on using the various security options.

Default File-Based Security Provider

The default security provider is provided as part of the Style Intelligence installation. It makes use of the following XML files located in the SREE Home directory (`webapps/sree/WEB-INF/classes`):

- `password.xml`: Stores users and groups.
- `role.xml`: Stores roles.
- `acl.xml`: Stores security permissions.

Use the default file-based security provider in the following circumstances:

1. You do not have any existing security data store that you want to reuse for your reporting application.

2. You have an existing security data store, but it is not feasible to connect to this store via the InetSoft custom security interface.

   In this case, you will need to duplicate all your users, groups, and roles for use within Style Intelligence, and set corresponding permissions. You can do this via Enterprise Manager and/or security API calls. You will be responsible for keeping your existing data store and the Style Intelligence security provider in sync.

3. You have implemented complete session-based SSO (see *Request Filter Example*), and your security is predominantly defined at the group/role level.

   In this case you do not need to define any users in the security provider. Simply define your groups/roles and set corresponding permissions. You can do this via Enterprise Manager and/or security API calls.

Next Step: *Security API: Configuring the File Security Provider*.

LDAP Security

Use the LDAP File Security Provider if you have an LDAP data store (such as Active Directory), irrespective of the SSO mechanism that you select.

Next Step: *Configuring an LDAP Security Provider* in *Advanced Server Administration*.

Custom DB-based or API-Based Security Provider

Implement a custom security provider if your setup requires the security provider to perform authentication of user credentials or user role/group mappings against an *external source*.

For example, if your security data store is a database or another security framework and you have form-based or partial session-based SSO, use a custom security provider. The best approach is the *composite security provider*, which splits the responsibility for authentication and authorization tasks. To do this, implement your own authentication provider and use the built-in file-based authorization provider.

Next Step: *Custom Security Integration* in *Advanced Server Administration*.

See Also

Security, in *Advanced Server Administration*, for information on configuring security in Enterprise Manager.

2 Java API Utility Applications

The following sections provide some helpful code snippets for common integration tasks using the public Java API.

Note that you can accomplish some of these tasks in an easier manner using the built-in tools provided by Style Studio and Enterprise Manager. Please consult the appropriate sections in *Advanced Report Design* and *Advanced Server Administration* for information on the preferred approaches (not requiring programming). In general, use the Java API only when these built-in tools do not address your particular needs.

2.1 Security API: Configuring the File Security Provider

If you are using the built-in security provider (see *Configuring the Default Security Provider*) and find it cumbersome to manually setup users, groups, roles, and permissions, you can programmatically configure the security provider using the security API. For LDAP and custom security implementations, use the interface provided by the external security data store to add and remove users, groups, and roles. The security provider should only pull from, and never push to, that store.

The code snippet below performs the following tasks:

- Create a new group of users called “Accounts”.
- Add a new user called “brian” to the “Accounts” group.
- Give the “Accounts” group read permissions on a folder called “Audit Reports”.

```
import inetsoft.sree.security.*;
import inetsoft.sree.*;
...

System.setProperty("sree.home", "{Absolute Path to your sree
home location}");

SecurityEngine sEngine = SecurityEngine.getSecurity();

FileSecurityProvider fsProvider =
(FileSecurityProvider)sEngine.getSecurityProvider();

// Get a handle to the authentication module
FileAuthenticationProvider fap =
(FileAuthenticationProvider)fsProvider.getAuthenticationPro
vider();

// Add the new group called 'Accounts'
fap.addGroup(new FSGroup("Accounts"));

// Add a user called 'brian' and assign him to the
'Accounts' group
FSUser user = new FSUser("brian");
```

```
user.setGroups(new String[] {"Accounts"});
fap.addUser(user);

// Give the Accounts group read permissions on a folder
called 'Audit Reports'

// Get a handle to the authorization module
FileAuthorizationProvider fathp =
(FileAuthorizationProvider)fsProvider.getAuthorizationProvi
der();

// Create a permission object
Permission folderPermission = new Permission();
folderPermission.setReadGroups(new String[] {"Accounts"});

// Set permissions on the folder
fathp.setPermission("Audit Reports", folderPermission);
```

2.2 Creating a Custom Report List

The section illustrates how to use the Style Intelligence API to compile a list of reports and Viewsheets for a given user. You can use this module in your own server pages. However, before you invest resources in creating a custom report list, consider simply embedding the contents of the default User Portal's **Report** tab within your application. This is the recommended and simplest implementation. See *Integrating Style Intelligence into Your Web Application* for further details.

The code snippet below presents the generated list of reports and Viewsheets in a Java List object. Each value in the List is an array containing the display name and encoded request URI of the report or Viewsheet.

```
package com.inetsoft.demo;

import java.util.*;
import inetsoft.sree.RepositoryEntry;
import inetsoft.sree.AnalyticRepository;
import inetsoft.sree.security.SRPrincipal;
import inetsoft.sree.internal.SUtil;
import inetsoft.util.Tool;

public class RepositoryList {
  public RepositoryList(SRPrincipal principal,
AnalyticRepository repository) {
    this.principal = principal;
    this.repository = repository;
  }

  public List getFolderEntryPaths(String folderName) throws
Exception {
    List<String[]> entryPaths = new ArrayList<String[]>();
    String reportParam = "op=FrameReplet&name=";
    String archiveParam = "op=FrameReport&name=";
    String vsParam = "op=vs&path=/";

    RepositoryEntry[] repEntries =
repository.getRepositoryEntries(folderName, principal, "r",
RepositoryEntry.FOLDER|RepositoryEntry.REPLET|RepositoryEnt
ry.VIEWSHEET|RepositoryEntry.ARCHIVE);
```

```
    for(int i=0; i < repEntries.length; i++) {
      String entryPath = repEntries[i].getPath();

      if(repEntries[i].getType() == RepositoryEntry.REPLET)
{
        String encodedPath = reportParam +
Tool.encodeURL(entryPath);
entryPaths.add(new String[]{ entryPath, encodedPath});
      }
      else if(repEntries[i].getType() ==
RepositoryEntry.VIEWSHEET) {
        String encodedPath = vsParam +
Tool.encodeURL(entryPath);
        entryPaths.add(new String[]{ entryPath,
encodedPath});
      }
      else if(repEntries[i].getType() ==
RepositoryEntry.ARCHIVE) {
        String encodedPath = archiveParam +
        Tool.encodeURL(entryPath);
        entryPaths.add(new String[]{ entryPath,
encodedPath});
      }
      else if(repEntries[i].getType() ==
RepositoryEntry.FOLDER){
        entryPaths.addAll(getFolderEntryPaths(entryPath));
      }
    }
    return entryPaths;
  }
  private SRPrincipal principal;
  private AnalyticRepository repository;
}
```

The following JSP code uses the above `RepositoryList` class to create a custom portal interface. The repository tree is then displayed as a drop-down selection list.

```
<%@ page language="java" contentType="text/html" %>
<%@ page
import="com.inetsoft.demo.RepositoryList"
import ="java.util.*"
import
="inetsoft.sree.security.SRPrincipal,inetsoft.sree.internal
.SUtil"
import
="inetsoft.sree.RepletRepository,inetsoft.sree.AnalyticRepo
sitory"
%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=windows-1252">
<title>InetSoft Portal</title>

<script>
  function generateReport() {
    document.getElementById('rFrame').src = "/sree/
Reports?" + document.form1.userReportList.value;
  }
</script>
</head>
```

```
<body>
<p>InetSoft Portal</p>
<form name=form1 method="POST">
<select  size="1" name="userReportList" >
<%
  // Get a handle to Repository for user (assume a valid
session exists)
  SRPrincipal principal =
(SRPrincipal)session.getAttribute(RepletRepository.PRINCIPA
L_COOKIE);
  AnalyticRepository repository =
(AnalyticRepository)SUtil.getRepletRepository();
  RepositoryList rList = new RepositoryList(principal,
repository);
  List<String[]> paths = rList.getFolderEntryPaths("/");

  for(int i = 0; i < paths.size(); i++) {
    String[] pArray = (String[])paths.get(i);
    String displayName = pArray[0];
    String valueName = pArray[1];
%>
  <option value=<%=valueName%>>
  <%=displayName%>
  </option>
<%
  }
%>
</select>
<input type="button" value="Generate Report" name="B1"
onClick="generateReport();"></p>
</form>
<iframe id=rFrame src="about:blank" width=95% height=90%></
iframe>
</body>
</html>
```

2.3 Managing the Data Repository (datasource.xml)

When you implement a service-based solution, it is often necessary to dynamically manipulate the data repository (data sources, data models, etc.). You can use the public API to perform such common tasks as dynamically expanding a developer's data model (as the developer creates new database fields) or adding an additional tenant connection.

The following code snippet illustrates how to gain access to the sample 'Orders' data source in the repository and add an additional tenant connection. The example also expands the data model to add a new attribute to the 'Product' entity which maps to a particular database field.

```
import inetsoft.uql.XFactory;
import inetsoft.uql.XRepository;
import inetsoft.uql.jdbc.JDBCDataSource;
import inetsoft.uql.service.XEngine;
import inetsoft.uql.erm.XDataModel;
import inetsoft.uql.erm.XLogicalModel;
import inetsoft.uql.erm.XEntity;
import inetsoft.uql.erm.XAttribute;
import inetsoft.uql.schema.XSchema;
...
```

```
System.setProperty("sree.home", ""{Absolute Path to your
sree home location}");

// Get a handle to the data repository
XRepository dataRepository = XFactory.getRepository();

// Edit the data model in the Orders data source
XDataModel dataModel =
dataRepository.getDataModel("Orders");

XLogicalModel logicalModel =
dataModel.getLogicalModel("Order Model");
XEntity entity = logicalModel.getEntity("Product");

// Create an attribute to map to a field in the DB
XAttribute attribute = new XAttribute("Category Num",
"SA.PRODUCTS", "CATEGORY_ID", XSchema.INTEGER);

// Add this attribute to the 'Product' entity
entity.addAttribute(attribute);

// Add an additional tenant connection
JDBCDataSource tenantConnection = new JDBCDataSource();
tenantConnection.setName("Orders Client1");
tenantConnection.setURL("jdbc:derby:classpath:orders");
tenantConnection.setDriver("org.apache.derby.jdbc.EmbeddedD
river");
tenantConnection.setUser("SA");
tenantConnection.setPassword("");

// Get a handle to the default 'Orders' data source
JDBCDataSource ordersDataSource =
(JDBCDataSource)dataRepository.getDataSource("Orders");
ordersDataSource.addDatasource(tenantConnection);

// Finalize changes
dataRepository.updateDataModel(dataModel);
```

See Also

Data Models, in *Advanced Data Modeling*, for information on expanding a data model.

Design for Multi-Tenant Environment, for complete information about multi-tenancy.

2.4 Accessing the Servlet Context

In rare situations, a report intended to be run only in a web environments may need to access the servlet context or configuration parameters. To do this, use the `ServletRepository` class:

```
Servlet servlet = ServletRepository.getServlet();
ServletConfig config = servlet.getServletConfig();
ServletContext context = config.getServletContext();
```

However, there are several disadvantages of accessing the servlet in a replet class:

- If a report accesses the servlet instance, it can only be used in a web environment.
- The repository must be run locally in the servlet. It is impossible to use a remote repository server in conjunction with RMI.

HTTP Request and Response

When a report is running inside a servlet, the HTTP request and response objects are accessible as special parameters in the replet request. You can retrieve these from the HttpServiceRequest and HttpServiceResponse objects using the RepletRequest constants `SERVICE_REQUEST` and `SERVICE_RESPONSE`.

```
// param is the RepletRequest object passed to
// createReport()
HttpServiceRequest srvreq = (HttpServiceRequest)
param.getParameter(RepletRequest.SERVICE_REQUEST);
HttpServletRequest srvreq.getRequest();

HttpServiceResponse srvres = (HttpServiceResponse)
param.getParameter(RepletRequest.SERVICE_RESPONSE);
HttpServletResponse resp = srvres.getResponse();
```

These parameters are not accessible if the report is not running inside a servlet. The ability to access the HTTP request object is designed only to satisfy some special needs in the Web environment, such as access to the session object. It should not be used to access the report parameters.

2.5 Pooling Database Connections Programmatically

The built-in connection pooling mechanism is described in *Specifying Data Source Information*, in *Advanced Server Administration*. You can choose instead to supply your own database connection pooling mechanism by implementing the ConnectionPool interface.

See Also

Specifying Data Source Information, in *Advanced Server Administration*, to configure pooling in Enterprise Manager.

2.5.1 getConnection()

This method is called whenever a new connection is needed by the query engine. The pool should return the next available connection. If there are no more connections in the pool, this method should block until a connection is made available. The method syntax is shown below:

```
getConnection(XDataSource xds, Principal user);
```

The parameter '`xds`' denotes the JDBC data source. Parameter '`user`' is a Principal object that identifies the user for whom the connection is being retrieved.

2.5.2 releaseConnection()

This method is called by the query engine after a connection is no longer needed. This method should be coordinated with the `getConnection()` method so that if there is a `getConnection()` pending, this method will notify the waiting thread.

Example: Connection Pool

The following is a simple connection pool implementation. It creates ten connections at startup.

```
import java.sql.*;
import java.util.*;
import java.security.Principal;
import java.util.concurrent.*;
import inetsoft.uql.*;
import inetsoft.uql.jdbc.*;

public class TestConnectionPool implements ConnectionPool {

  public TestConnectionPool() {
    try {
       Class.forName("org.apache.derby.jdbc.EmbeddedDriver"
);
    }
    catch(ClassNotFoundException classx) {
    }

    String url = "jdbc:derby:classpath:orders";

    for(int i = 0; i < SIZE; i++) {
      try {
        Connection connection =
DriverManager.getConnection(url, "sa", "");
        connections.push(connection);
      }
      catch(SQLException sqlx) {
      }
    }
  }

  public Connection getConnection(XDataSource xds,
Principal user) {
    System.out.println("getConnection("+xds+", "+user+")");
    Connection conn = null;
    if(xds.getName().equals("Orders")) {
      try {
        conn = connections.takeFirst();
      }
      catch (InterruptedException ie) {
      }
    }
    return conn;
  }

  public void releaseConnection(XDataSource xds, Connection
conn) {
    System.out.println("releaseConnection("+xds+",
"+conn+")");
    connections.push(conn);
  }
```

```
  private final BlockingDeque<Connection> connections = new
LinkedBlockingDeque<Connection>(SIZE);

  private static final int SIZE = 10;
}
```

¢

2.5.3 Setting the Connection Pool

There are two methods for setting the connection pool. The first method is to set the property `jdbc.connection.pool` in *sree.properties*.

```
jdbc.connection.pool=SimpleConnectionPool
```

The second method is to programmatically call `JDBC.setConnectionPool()`.

```
ConnectionPool pool = new SimpleConnectionPool();
inetsoft.uql.jdbc.JDBCHandler.setConnectionPool(pool);
```

Note that it is important to call the `close()` function of a JDBCTableNode after using it, because if the system is busy it may cause a deadlock in the connection pool.

2.6 Scheduler API

Enterprise Manager allows you to specify built-in schedule conditions and actions. Built-in conditions include several time-based conditions: daily, weekly, monthly, a day in a week in a month (e.g. 1st Sunday in March), a day in a week in a year (e.g. 20th Wednesday of the year) and an exact date and time. Built-in actions include report generation, export, delivery, notification, printing, and archiving.

In addition to these default conditions and actions, you can also use the Java API to implement scheduled reports for custom conditions and actions.

See Also
Scheduler, in *Advanced Server Administration*, for full information on Scheduler operation.

2.6.1 User-Defined Conditions

A user-defined condition must implement the UserCondition interface. Two key methods in the interface are `check()` and `getRetryTime()`. Both methods take a current time parameter. The current time parameter specifies the logical current time. Its value may be different from the actual system time returned by `System.currentTimeMillis()` and is used for the 'what if' analysis by the Scheduler. Both routines should use the passed value as the current time.

The `getRetryTime()` method is called by the Scheduler at start-up to determine the time this condition needs to be checked. The method is also called after a `check()` call returns false. If a task has more than one condition, the task will be retried at the latest retry time of all the conditions. If the `getRetryTime()` method returns a -1, the condition is abandoned.

The `check()` method can be called anytime after the retry time. It should check the condition criteria and return true if all the criteria are satisfied and false otherwise. It is up to the user-defined class to perform any checking in the `check()` method.

Example: User-Defined Condition

The following example illustrates a user-defined condition that checks the existence of a file.

```
import inetsoft.sree.schedule.*;
import inetsoft.sree.*;
import java.io.*;

public class FileCondition implements UserCondition {

   public String getLabel() {
      return "File Check";
   }

   public RepletRequest getRepletRequest() {
      return this.request;
   }

   public void setRepletRequest(RepletRequest request) {
      this.request = request;
   }

   public long getRetryTime(long now) {
      // if the file name property is not defined by user
      if(filename == null) {
         return -1;
      }

      return now + 600000; // retry in 10 minutes
   }

   // check if the file exists
   public boolean check(long now) {
      return (new File(filename)).exists();
   }

   String filename = SreeEnv.getProperty("extra.datafile");
   RepletRequest request = null;
}
```

¢

See Also

Scheduler Conditions, in *Advanced Server Administration*, to set a built-in or custom condition in Enterprise Manager.

2.6.2 User-Defined Actions

In addition to the built-in Schedule actions, you can also define custom actions. A user-defined action class must implement the UserAction interface. There are three methods in the interface.

- The `run()` method is the main routine for performing the action.
- The other two methods, `setRepletRequest()` and `getRepletRequest()`, deal with the parameters for the user action.

You can configure a set of parameter values when you create an action in Enterprise Manager. The parameters are loaded and passed to the action through the `setRepletRequest()` method. This allows a user-defined action class to be parameterized in the same way as the default actions.

The DefaultUserAction class provides a default implementation of the UserAction interface. The subclass of DefaultUserAction only needs to define the `run()` method. As with conditions, the `equals()` method should also be implemented.

To make the user-defined action available for selection on the Enterprise Manager's **Action** tab (for a given scheduled task), add the `replet.viewer.schedule.actions` entry into *sree.propertes*. See *User-Defined Action* in *Advanced Server Administration* for more information, and the example below.

Example: User Action

The class below specifies a user-defined action called "MyAction." The action executes a deployed report named "SomeReplet," and exports the report to Excel format as `C:\\SomeReplet.xls`.

```
import inetsoft.sree.schedule.*;
import inetsoft.sree.*;
import inetsoft.sree.security.*;
import inetsoft.report.io.*;

import java.util.*;
import java.security.*;
import java.io.*;

public class MyAction extends DefaultUserAction implements
RepletSupport {

  public MyAction() {
    super();
  }

  public MyAction(String replet) {
    this.replet = replet;
  }

  public void run(Principal principal) throws Throwable {
    FileOutputStream fos = null;

    try {
```

```
      final RepletRepository rr =
SreeEnv.getRepletRepository();

      final Object repletId = rr.createReplet(replet,
principal, RepletRepository.REPLET);

      RepletRequest request = getRepletRequest();
      /*
        ...
        Modify parameters here.
        ...
      */

      rr.generateReplet(repletId, request);

      final Object exportId = rr.export(repletId,
Builder.EXCEL_DATA);

      fos = new FileOutputStream("C:\\"+replet+".xls");

      byte[] b = null;
      while((b = rr.nextBlock(exportId)) != null) {
        fos.write(b);
      }

      fos.flush();
    }
    catch(Exception e) {
      e.printStackTrace();
      throw new RepletException(e);
    }
    finally {
      try {
        fos.close();
      }
      catch(Exception e) {}
    }
  }

  public String getRepletName() {
    return replet;
  }

  public void setRepletName(String replet) {
    this.replet = replet;
  }

  public String getLabel() {
    return "MyAction";
  }

  private String replet = "SomeReplet";
}
```

To make the “MyAction” user-defined action available for selection in the Enterprise Manager’s **Action** tab (for a given scheduled task), add the following line into the *sree.propertes* file.

```
replet.viewer.schedule.actions=MyAction
```

See Also

Scheduler Actions, in *Advanced Server Administration*, to set a built-in or custom action in Enterprise Manager.

2.7 Presenters

A presenter is an object that draws a graphical representation of a numerical value. (See *Cell Presenter* in *Advanced Report Design*.) Typically, you assign a presenter to a particular column of a table. However, you can also associate a presenter with an object type, in which case it applies to all objects of that type in a document.

The presenter is a powerful tool that allows you to broadly extend Style Intelligence by rendering any user-defined object value. It has a very simple set of methods, which are defined in the inetsoft.report.Presenter interface. See the API JavaDocs for full details.

Walkthrough

The Style Intelligence package comes with two sample presenters that display a numeric value as a horizontal bar. In the following example you will build a slightly different bar presenter. Instead of a bar growing from left to right, you will create a presenter with bars growing from right to left.

You will first define a few constructors, a default constructor and a constructor that takes a maximum value and color as parameters. The maximum value is used to calculate the size of the bar. The size is the proportion of the actual value to the maximum value.

`RBarPresenter.java is found in the {InetSoftInstallation}/examples/docExamples/core directory.`

```
public class RBarPresenter implements Presenter {

   public RBarPresenter() {
   }

   public RBarPresenter(double max, Color color) {
      this.max = max;
      this.color = color;
   }
```

Pick a default size for the bar presenter and allow the size to be changed by users:

```
public Dimension getPreferredSize(Object v) {
   return psize;
}

public void setPreferredSize(Dimension psize) {
   psize = new Dimension(psize);
}
```

You also need to define the `isPresenterOf()` method to check if a type of object can be presented by this presenter. This is used when a presenter is

assigned to a table column to avoid using it on the wrong types of objects, if there is more than one type in the column.

```
public boolean isPresenterOf(Class type) {
   return Number.class.isAssignableFrom(type);
}
```

Last, define the `paint()` method for actually painting the bar. You can calculate the width of the bar using the value passed into `paint()` and the maximum value and presenter area width. Then align the bar to the right of the area and paint.

```
public void paint(Graphics g, Object v, int x, int y, int w,
int h) {
   if(v != null && v instanceof Number) {
      Rectangle clip = g.getClipBounds();
      Color oc = g.getColor();
      g.setClip(x, y, w, h);
      double n = ((Number) v).doubleValue();
      g.setColor(color);
      int width = (int) (n * w / max);
      g.fillRect(x+w-width, y+2, width, h-4);
      g.setColor(oc);
      g.setClip(clip);
   }
}
```

Figure 3. Report using RbarPresenter (PresenterEx.java)

Company	Close	Tracked	Rating	Date
ADE CORPORA	18.38	true		1998-04-07 00:0
AEHR TEST SYS	3.63	false		1996-02-11 00:0
AG ASSOCIATES	2.75	false		1998-07-23 00:0
ALIGN-RITE INTI	10.06	false		1997-11-07 00:0
AMISTAR CORP	2.38	false		1998-04-21 00:0
APPLIED MATER	24.5	true		1995-01-05 00:0
APPLIED SCIEN	4	true		1998-09-20 00:0
ASECO CORPOI	1.72	false		1995-06-17 00:0
ASM INTERNAT I	4.75	false		1995-03-13 00:0
ASYST TECHNO	7.81	true		1994-09-05 00:0
ATMI INC	14.44	false		1993-05-11 00:0
BE SEMICONDU	6.13	false		1997-07-15 00:0
BROOKS AUTOM	8.88	true		1997-09-08 00:0
BTU INTERNATI	2.63	false		1998-07-12 00:0
CFM TECHNOLO	8	false		1995-06-25 00:0
COHU INC	17.75	false		1996-02-11 00:0
CREDENCE SYS	14.44	false		1994-01-17 00:0
CYMER INC	10	true		1998-09-20 00:0
DUPONT PHOT	23.75	false		1996-07-23 00:0
ELECTROGLAS	9.25	false		1998-04-06 00:0
ESTERLINE TEC	18.13	false		1995-12-16 00:0
ETEC SYSTEMS	30	true		1998-08-20 00:0
FERROFLUIDIC	3.63	false		1996-09-17 00:0
FSI INTERNATIO	6.06	false		1998-04-03 00:0
FUSION SYSTEM	0.06	false		1996-06-15 00:0
GASONIC INTER	3.81	true		1998-05-23 00:0
GENERAL SCAN	6	false		1994-11-17 00:0
GENUS INC	0.88	false		1996-02-11 00:0
HELIX TECHNOI	9.94	true		1992-09-10 00:0
IFR SYSTEMS IN	4.38	false		1997-06-17 00:0
INTEGRATED MI	5.5	true		1998-08-21 00:0
KEITHLEY INSTF	5.13	false		1995-06-24 00:0
KLA-TENCOR C	22.38	false		1998-07-18 00:0
KULICKE & SOF	13.44	false		1998-07-23 00:0
LAM RESEARCH	10.06	true		1998-07-11 00:0
LTX CORP	1.81	false		1995-05-27 00:0
MATTSON TECH	3.25	false		1996-06-04 00:0

2.8 Exporting Reports Programmatically

The following sections explain how to export reports to various formats using the Java API.

See Also

Exporting a Report, in *Advanced Report Design*, to export a report from Style Studio.

Scheduler Actions, in *Advanced Server Administration*, to schedule a report export.

Exporting a Report, for information on exporting from the User Portal.

2.8.1 Exporting to PDF Programmatically

Portable Document Format (PDF) is a document format created by Adobe for distributing documents online. It has the advantages of compact size and the availability of a free document viewer (Acrobat Reader) on many platforms.

Style Intelligence provides support for PDF printing either through the use of the PDF3Generator or through the use of the PDF4Generator. `PDF3Generator.getPDFGenerator()` returns an appropriate object for the user's environment. Therefore, you should use `PDF3Generator.getPDFGenerator()` to get an instance of the above classes, rather than directly instantiating the constructors.

Printing to a PDF File

The snippet below illustrates how to use the PDF3Generator or PDF4Generator object to print to a PDF file.

```
...
try {
   ReportSheet report = createReport();

   FileOutputStream output =
      new FileOutputStream("output.pdf");
   PDF3Generator pdf =
      PDF3Generator.getPDFGenerator(output);
   pdf.generate(report);

   PreviewView previewer = Previewer.createPreviewer();
   previewer.setExitOnClose(true);

   previewer.pack();
   previewer.setVisible(true);

   previewer.print(report);
   String prop = ReportEnv.getProperty("print");
   if(prop != null && prop.equals("true")) {
      previewer.printAction();
   }
} catch(Exception e) {
   e.printStackTrace();
}
```

```
public static ReportSheet createReport() {
   try {
      FileInputStream input =
         new FileInputStream("report.srt");
      Builder builder =
         Builder.getBuilder(Builder.TEMPLATE, input);
      return builder.read(".");
   } catch(Exception e) {
      e.printStackTrace();
   }

   return null;
}
```

The PDFPrinter is a simpler version of the PDF3Generator. The PDFPrinter lacks support for generating tables and TOC (Table of Contents). To create a PDFPrinter, pass a File object or a FileOutputStream to the constructor:

```
PDFPrinter pdf = new PDFPrinter(new File("report.pdf"));
```

Once you have created the PDFPrinter object, you can retrieve a PrintJob from it by calling `getPrintJob()`:

```
report.print(pdf.getPrintJob());
```

After printing is finished, close the PDF stream:

```
pdf.close(); // or pdf.getPrintJob().end();
```

It is important to close the PDF stream after printing, otherwise the PDF file may not be complete. You can close the PDF output either by calling the `end()` method on the PrintJob object returned by the PDFPrinter, or by calling the `close()` method on the PDFPrinter directly.

Compression Options

By default, Style Intelligence compresses text and images when exporting to PDF format. On CJK operating systems, however, compression should be disabled in order to prevent corruption of the of the binary data. To turn off compression, use the following commands:

```
pdf.setCompressText(false);
pdf.setCompressImage(false);
```

Note however, that the resulting uncompressed PDF file can be quite large, especially if you use images or painters in the report.

Font Handling in PDF (Base-14 Fonts)

PDF viewer software guarantees that the following fonts (the “base14 fonts”) will be available for display purposes:

– courier

– courier-bold
– courier-boldoblique
– courier-oblique
– helvetica
– helvetica-bold
– helvetica-boldoblique
– helvetica-oblique
– symbol
– times-bold
– times-bolditalic
– times-italic
– times-roman
– zapfdingbats

To ensure maximum portability, Java font objects are mapped into the base14 fonts when generating PDF files.

Table 1. Default mapping between Java Fonts and Base-14 Fonts

JAVA FONT NAME	BASE14 FONT NAME
Dialog	Helvetica
Dialoginput	Courier
Serif	Times
SansSerif	Helvetica
Monospaced	Courier

You can change this mapping as follows:

```
pdf.putFontName("dialog", "Times");
```

By default the PDFPrinter assumes the output page is letter size (8.5 x 11). To change this, set the page size property:

```
// switch width and height to print in landscape
pdf.setPageSize(11, 8.5);
// or print on A4 paper
pdf.setPageSize(StyleConstants.PAPER_A4);
```

Embedded Fonts

The PDFPrinter is the basic implementation for generating PDF files. It only supports the base-14 fonts defined by the PDF standard. Because the base-14 fonts are guaranteed to exist in all PDF viewers, the generated files are very portable and compact. However, there are situations where you may need other fonts in order to meet formatting requirements.

For example, if the PDF document is generated using Unicode characters greater than 256, it is a good idea to embed the fonts.

An enhanced PDF generation class, PDF3Printer, supports embedded fonts in the PDF file. Since the resulting PDF files use the same fonts as the reports, they more accurately reflect the report presentation. To support font

embedding, the PDF generator needs to access the TrueType font files on the local file system. It parses the font file for font information otherwise not available from the Java API and loads the font data to embed into the PDF file.

The TrueType font directories need to be specified using the `font.truetype.path` property. The property is a directory path, e.g., *c:\winnt\fonts*, and can contain multiple directories separated by a path separator (semicolon on Windows and colon on Unix). Only TrueType fonts on this path are used in PDF generation.

```
ReportEnv.setProperty("font.truetype.path", "c:/winnt/
fonts;c:/otherfonts");
```

Type 1 font information is retrieved from AFM files. AFM is the standard font format used by Adobe to store font data. AFM files can be downloaded from the Adobe Web site. The PDF generator uses `font.afm.path` to search AFM files for a Type 1 font. Applications using Type 1 fonts need to package the AFM files with the application and set the `font.afm.path` to point to the AFM directory.

```
import inetsoft.report.pdf.*;
...
FileOutputStream pdffile = …;
ReportEnv.setProperty("font.truetype.path", "c:/winnt/
fonts");
PDF3Printer pdf = new PDF3Printer(pdffile);
report.print(pdf.getPrintJob());
pdf.close();
```

The PDF3Printer class is derived from the PDFPrinter class and shares the same methods for controlling PDF generation, such as image and text compression. It has additional methods for controlling the font embedding. By default, only font meta-data is embedded in the PDF files. For TrueType fonts, the entire font file can be embedded in the PDF file. This provides maximum portability at the price of larger file size.

```
pdf.setEmbedFont(true); // embed font file
```

If a font's file is not found on the specified font path, it is mapped to a base-14 Font in the same way as PDFPrinter.

PDF Bookmarks and Hyperlinks

PDF format supports bookmarks. Bookmarks are basically a hyperlinked Table of Contents displayed in a separate view than the document. Users can use the bookmark items to navigate the document. To produce a PDF files with bookmarks, use PDF3Generator.

```
import inetsoft.report.pdf.*;
...
FileOutputStream pdffile = …;
```

```
PDF3Generator pdf =
   PDF3Generator.getPDFGenerator(pdffile);
ReportSheet report = …;
pdf.generate(report);
// no close() necessary
```

To prevent the generation of bookmarks entirely, use the PDF3Printer instead of PDF3Generator.

If the report contains heading elements, including heading elements in subreports, these are converted into PDF bookmarks. To ignore heading elements in subreports, set the `selectiveBookmarks` property to `true`.

```
PDF3Generator.setSelectiveBookmarks(true);
boolean selective = PDF3Generator.isSelectiveBookmarks();
```

To initialize PDF bookmarks in their "open" state, set the `pdf.open.bookmark` property to `true` by using the `setOpenBookmark()` method of PDFPrinter. To check the current value of the `pdf.open.bookmark` property, use the `isOpenBookmark()` method of PDFPrinter.

```
PDFPrinter.setOpenBookmark(true);
boolean openBookmark = PDFPrinter.isOpenBookmark();
```

Similarly, to initialize PDF thumbnails in their "open" state, set the `pdf.open.thumbnail` property to `true` by using the `setOpenThumbnail()` method of PDFPrinter. To check the current value of the `pdf.open.thumbnail`, use the `isOpenThumbnail()` method of PDFPrinter.

```
PDFPrinter.setOpenThumbnail(true);
boolean openThumbnail = PDFPrinter.isOpenThumbnail();
```

Note that the `pdf.open.bookmark` and `pdf.open.thumbnail` properties are mutually exclusive. In the case where both are set to true, the `pdf.open.bookmark` setting will take priority.

The PDF document can also reproduce hyperlinks contained the report. A report property, `pdf.generate.links`, controls whether the hyperlinks will be generated. This property is set to true by default. If set to false, the generation time of the file will be decreased.

CJK Characters in PDF

The PDF 1.3 specification (Acrobat 4 and higher) supports CID fonts, which permit representation of 16-bit Chinese-Japanese-Korean (CJK) characters.

Style Intelligence provides two classes to support CID fonts: PDF4Printer and PDF4Generator. These correspond to the PDF3Printer and PDF3Generator classes for non-CID fonts. Output from PDF4Printer and PDF4Generator can only be viewed with Acrobat 4 and higher.

Installing Fonts and Inserting CJK Text

To generate documents with CJK fonts, install the Adobe Acrobat Asian Language Pack on your machine. You can choose to install one or all of the three languages supported by the pack: Chinese, Japanese, and Korean. The language pack installs CID fonts for each of the selected languages.

CID fonts use the same format as TrueType fonts. However, the internal tables in the fonts are slightly different from regular TrueType fonts. Therefore, the CID fonts installed by the language pack cannot be used in Java (or OS) as regular fonts.

In order to create a CJK document, you must also install a regular TrueType font for the language on the machine. The following example uses Japanese to illustrate the process. Chinese and Korean fonts are handled in the same way.

To create a document containing Japanese characters, first select the correct Japanese TrueType font (OS font) for the report element:

```
// "MS Mincho" is a TrueType Japanese font
report.setCurrentFont(new Font("MS Mincho", Font.PLAIN, 8));
```

Use Style Studio to enter the desired Japanese characters into the report file. (Reports use UTF8 encoding for text values, and can correctly encode all unicode characters.)

Creating PDF4Printer or PDF4Generator

After the report is created, you can preview it as usual. To generate the PDF document from the report, create a PDF4Printer (or PDF4Generator, which handles PDF bookmarks described earlier).

```
FileOutputStream output = new
FileOutputStream("output.pdf");
ReportEnv.setProperty("font.truetype.path",
   "c:/winnt/fonts;d:/usr/acrobat4/resource/cidfont");
PDF4Printer pdf = new PDF4Printer(output);
pdf.putFontName("MS Mincho", "HeiseiMin-W3-Acro");
```

Specifying the CID Font Directory

Use the `font.truetype.path` property to specify the directory where the Asian font pack is installed (which you can select during the installation process). This is necessary so that the PDF4Printer can find the CID font files and extract font information.

The PDF4Printer uses CMap files in addition to CID fonts. Since the CMap files are installed under the CMap directory parallel to the *cidfont* directory, the font loading program checks the '`../CMap`' directory using the

`font.truetype.path` directories. If the CID fonts are installed differently and the CMap files are placed at a different location, the CMap directory can be specified using the `font.cmap.path` property.

Creating a Mapping from TrueType to CID

After you create the PDF4Printer, you need to create a mapping from the TrueType Japanese font to the CID Japanese font. The Adobe Asian Language Pack comes with two Japanese fonts, HeiseiMin-W3-Acro and HeiseiKakuGo-W5-Acro. Either can be used in the mapping.

The following is a complete list of CJK fonts that are supported:

– HeiseiKakuGo-W5-Acro
– HeiseiMin-W3-Acro
– HYGoThic-Medium-Acro
– HYSMyeongJo-Medium-Acro
– HYSMyeongJoStd-Medium-Acro
– KozMinPro-Regular-Acro
– MHei-Medium-Acro
– MSung-Light-Acro
– MSungStd-Light-Acro
– STSong-Light-Acro
– STSongStd-Light-Acro

When the mapping has been properly configured, you can generate the PDF document as usual:

```
report.print(pdf.getPrintJob());
pdf.close();
```

CID fonts can be embedded in the generated document just like other TrueType fonts. Each CID font also has an additional CMap file. To embed the fonts, it is recommended that the CMap files also be embedded. Otherwise, the Asian language pack has to be installed on the viewer machine before the PDF document can be viewed.

```
pdf.setEmbedFont(true);
pdf.setEmbedCMap(true);
```

Note that for readers who are already CJK-enabled, `setEmbedCMap` should be set to `false`.

```
pdf.setEmbedCMap(false);
```

Because each CID font contains thousands of glyphs, embedding the entire font file will result in very large output files. Therefore, the font embedding in Style Intelligence extracts only the glyphs used in the current document and discards all other data to produce optimum size files. This process is more expensive than regular file embedding; therefore, generating CJK

PDFs with font embedding will be slower than generating regular PDF files.

2.8.2 Exporting to Excel Format Programmatically

Style Intelligence allows you to export a report to Excel format. The generated Excel file matches the report layout and formatting as closely as possible, and contains the actual data in the report so it can be further manipulated after export. However, certain formatting characteristics of the original report will not be retained in the Excel file. For example, page margin settings created in Style Studio are not available in the exported Excel file.

For the “Excel (no pagination)” and “Excel (best data editing)” formats, you can export supported formulas as Excel tables. Style Intelligence supports GroupFilter and TableSummaryFilter for export in Excel format, while NthMostFrequent and WeightedAverage formulas are not supported.

ExcelGenerator and ExcelSheetGenerator Classes

To export a report to an Excel file, use the ExcelGenerator and ExcelSheetGenerator classes.

```
ExcelGenerator gen =
   ExcelGenerator.createExcelGenerator(
      new FileOutputStream("report.xls"));
gen.generate(report);
```

The default page size is the standard letter size (8.5"x11"). To change the page size, call `setPageSize()` in ReportSheet before you invoke the `generate()` method.

The report length should not exceed the limits of the Excel spreadsheet. Also,

you may need to increase the heap size of the JVM. Use `java -XmxbbbM` to indicate the maximum (where maximum JVM size is `Xmx` and JVM heap size is `bbbM`). For example:

```
java -Xmx192M TestCase
```

When you export a CSV file to Excel, special characters are encoded in UTF8 format, which might provide an inaccurate representation of the report. In this case, set the `text.encoding.utf8` property in *sree.properties* to `false` to ensure that special characters are not encoded into UTF8 during export. To insert a form feed at every page break, set `text.break.pages` to `true`.

Different Excel Versions

You can export to formats suitable for particular versions of Excel, such as Excel 2000, Excel 95, or an Excel Spreadsheet with no page breaks for easier data manipulation. Instantiate the ExcelGenerator with the version parameter as follows:

```
//Excel 2000
ExcelGenerator gen =
   ExcelGenerator.createExcelGenerator(
      new FileOutputStream("report.xls"),
      ExcelGenerator.EXCEL2000);

//Excel 95
ExcelGenerator gen =
   ExcelGenerator.createExcelGenerator(
      new FileOutputStream("report.xls"),
      ExcelGenerator.EXCEL95);

//Excel with no page breaks
ExcelSheetGenerator gen =
   ExcelSheetGenerator.createExcelGenerator(
      new FileOutputStream("report.xls"),
      ExcelGenerator.EXCEL_SHEET);
```

2.8.3 Exporting to HTML Programmatically

HTMLGenerator is used to export a report in HTML format. Because HTML does not have many of the formatting capabilities of Style Intelligence, the generated HTML may omit some formatting information that is present in the original report.

As part of the HTML generation process, the builder may need to generate additional files that are used inside the HTML file. This is true for any non-text elements, such as charts, images, etc.

```
FileOutputStream os = new FileOutputStream(filename);
Builder builder = Builder.getBuilder(Builder.HTML, os, ".");
builder.write(sheet);
os.close();
```

While the Builder can be used to get the appropriate Builder for HTML generation, the HTMLGenerator performs the generation and is found in the `inetsoft.report.io` package.

Image Files

The extra parameter in the `getBuilder()` call specifies the directory in which to place the additional image files. If it is omitted, it defaults to the current directory. Image files are produced as PNG or JPEG files.

```
ReportEnv.setProperty("image.type", "png");
//or
ReportEnv.setProperty("image.type", "jpeg");
```

If an image element in the report uses a URL for image location, no image file will be generated. Instead, the specified URL is placed in the generated HTML as the image location.

Exporting an HTML Bundle

You can export an HTML Bundle, which is a ZIP file containing the generated HTML report and all associated image files. To generate an HTML Bundle, first create an OutputStream to be used by the Builder, and get the Builder to generate the HTML Bundle.

```
FileOutputStream os = new FileOutputStream(filename);
Builder builder =
   Builder.getBuilder(Builder.HTML_BUNDLE, os, ".");
builder.write(sheet);
```

The HTML Bundle is generated by the Builder class using `inetsoft.report.io.HTMLFormatter`. The Builder seamlessly handles writing to the ZipOutputStream.

Setting the property on ReportEnv as described above also controls the type of image file, JPEG or PNG.

Setting the DHTML Meta-Data Information

You can set the DHTML meta-data before exporting the report to HTML. The following properties are available:

```
report.title
report.subject
report.author
report.keywords
report.comments
report.created
report.modified
```

You can set these in the following way:

```
ReportSheet report = new TabularSheet();
report.setProperty("report.title", "Hello");
```

2.8.4 Exporting to RTF Programmatically

The Rich Text Format (RTF) export format has more limited layout options that Style Intelligence. Therefore, the generated RTF file will not always capture the complete formatting of a report. However, all report data, including text, table and image (painters), is saved in the RTF file. Because RTF exports are most often used for purposes of data manipulation rather than for presentation, the loss of formatting information is not critical in most instances. If you require the exported file to preserve formatting with high fidelity, use the PDF or Postscript export formats.

To create an RTF file, obtain a `Builder` object for the RTF export and then write the report to the output file:

```
FileOutputStream os = new FileOutputStream(filename);
Builder builder = Builder.getBuilder(Builder.RTF, os);
builder.write(sheet);
os.close();
```

You can also use `Builder.RTF_LAYOUT` to get the Builder object. This format conforms more precisely to the layout of the report. However, since elements are placed in blocks in the generated file, they may be harder to edit in some instances.

2.8.5 Exporting to Text File (CSV) Programmatically

The CSV export feature saves the report's tabular data into a text file, delimited by commas (default). This allows you to import the exported data into another program (e.g., spreadsheet) to make further manipulations. Multiple report tables are exported to the CSV file one-by-one.

The CSV formatter is included in the `inetsoft.report.io` package.

```
FileOutputStream os = new FileOutputStream(filename);
Builder builder = Builder.getBuilder(Builder.CSV, os);
builder.write(sheet);
os.close();
```

Use the *DelimitedFormatter* class directly to change the delimiter character. The following code segment changes the delimiter to a tab and quotes each field with a double quote.

```
FileOutputStream os = new FileOutputStream(filename);
DelimitedFormatter fmt = new DelimitedFormatter(os);
fmt.setDelimiter("\t");
fmt.setQuote("\"");
Builder builder = new Builder(fmt);
builder.write(sheet);
os.close();
```

2.9 Portal Customization

The following sections discuss customization of the User Portal and report viewing.

2.9.1 Custom Buttons

You can add a custom button to the report toolbar by using a ViewerAction. A ViewerAction object specifies the icon and label for the button and the action to take when the button is pressed. The *ViewerAction* interface in *inetsoft.sree* package defines the API for the custom button object.

Table 2. ViewerAction methods

getIconResource()	Return the icon resource name. The icon will be loaded as the specified resource and used to create a button on the viewer.
getLabel()	Return the label for the button. If the viewer is configured to use a text-only toolbar on a browser, the label is used to create a text based tool button. Otherwise, the label is used as the tool tip for the tool button.
actionPerformed()	Handles button click action. It can perform application specific actions when the button is pressed.

Custom Button Example

Note that the default toolbar already contains a 'Refresh' button.

This example explains how to create a custom refresh button. To create a button, first create a class that implements the ViewerAction interface to define the custom button. Create a graphic icon to represent the button on the viewer, and place the icon on the classpath so that it can be accessed as a resource.

```
public String getIconResource() {
   return "/images/dbRefresh.gif";
}
```

This means the GIF file should be placed in a directory called *images* on the classpath, or in a JAR file.

Next, define the label for the button.

```
public String getLabel() {
   return "Refresh DB";
}
```

Finally, define the `actionPerformed()` method, which accepts four parameters:

- replet ID
- replet repository instance
- replet object where the action is invoked
- repletRequest object

The method is free to perform any actions and can return a RepletCommand to direct the viewer to execute one or more viewer actions.

```
public RepletCommand actionPerformed(Object rid,
   RepletRepository repo, Replet replet, RepletRequest req)
{
  // actions, such as refreshing the database data

  return RepletCommand.refresh(); // refresh report
}
```

Configuring a Custom Button

Once the custom button class is created, you need to be registered it with the viewer so it will be added to the toolbar. To register the button, add a `replet.viewer.actions` property to the *sree.properties* file. The value for the property is the fully qualified class name of the custom button class.

```
replet.viewer.actions=com.mycompany.buttons.DBRefresh
```

If more than one button needs to be added, use a comma-separated list.

2.9.2 Custom Repository Protocol

The Style Intelligence report server uses one of the following protocols: RMI, HTTP, or SOAP. You can also implement a repository proxy class to provide support for a different protocol. This third-party protocol support is intended mainly for communications between a custom viewer and the report server. For a servlet-based viewer, as the repository already resides on the server machine, there is no need to use a custom protocol to communicate between the servlet and repository.

To incorporate a different protocol, follow these steps:

1. Create a Repository Client Proxy.

   The proxy should implement the RepletRepository interface. In each method, it should forward the request (method parameters) to the server and pass the result back.

   ```
   public class RepositoryProxy implements RepletRepository {
     public RepositoryProxy(...) {
       // establish connection to the server
       // using the third-party protocol
     }

     public Object create(String name, Object ticket) throws
   RemoteException, RepletException {
       // call the server to perform a create()
       // return the result from the server
     }
   }
   ```

2. Create a Repository Server.

   The server exposes methods to be called by the proxy using the same protocol. It can use the Replet Engine as the implementation of the repository.

   ```
   public RepositoryServer2 {
     public RepositoryServer2() {
       engine = new RepletEngine();
       engine.init();
       // register with protocol as needed...
     }
   ```

```
  // The signature of this method is protocol-dependent.
  // We use the same signature as the RepletRepository for
  // convenience. This is the method the client proxy
  // calls through the protocol
  public Object create(String name, Object ticket) {
    return engine.create(name, ticket);
  }
}
```

3. Create a proxy repository in the client and pass it to the viewer:

```
RepositoryProxy proxy = new RepositoryProxy(...);
Viewer viewer = new Viewer(proxy);
```

Depending on the protocol selected for the proxy and server, the implementation needs to convert the Java object parameters and return values to the data structures supported by the protocol. All objects used in the RepletRepository methods are serializable. Therefore, the easiest way to convert the parameters is to serialize the objects into bytes, pass them across the protocol as raw data and convert them back to Java objects at the receiving end.

3 SOAP Web Services

Style Intelligence provides a number of server side reporting functions that are accessible as web services. These web services enable clients which may be running on different operating systems and using different programming languages to access server side enterprise reporting tools through remote procedure calls. There are two web services provided:

Note: As of version 10.2, the SOAP engine provided with the Style Intelligence installation has been changed from Axis2 to Metro. (http://metro.dev.java.net) The fundamental API remains the same, but a number of small changes to the WSDL to accommodate compatibility with Java 1.6 and stricter standards prevents backwards compatibility with SOAP clients written for earlier versions.

- **SoapRepository** – This web service facilitates execution, archiving and scheduling of reports deployed in the report repository.
- **SoapDataService** – This web service facilitates execution of queries and worksheet assets defined in Style Studio.

3.1 Web Service Deployment

The SOAP web services provided by Style Intelligence can be deployed with native support for WebLogic and WebSphere. The same web services can also be deployed into other J2EE compliant web application servers by using the Axis2 SOAP implementation. Instructions on deployment for each SOAP engine are provided below.

3.1.1 Apache (Axis2)

Axis2 is a servlet-based SOAP implementation from the Apache Group. The SOAP engine is deployed as a regular HTTP servlet. Since Axis2 is a generic servlet, it can be deployed on any J2EE compliant web server.

Archive the necessary files into a WAR file using the `jar` command. The structure for the WAR file should follow the directory structure below:

- *WEB-INF/conf*: *axis2.xml*
- *WEB-INF/services*: Include the directory for each service you wish to deploy.
- *WEB-INF/classes*: *repository.xml*, *sree.properties*, *datasource.xml*, *query.xml*, and other XML files in the home directory
- *WEB-INF/lib*: *etools.jar*, *bisuite.jar*/*sree.jar*, and other JARs required for report execution

Note: The ‘SoapRepository’, ‘SoapDataService’, and ‘PeopleService’ web services are pre-deployed with the default installation.

You should update the `axis2.xml.path` and `axis2.repository.path` parameters in your *web.xml* file.

3.1.2 WebLogic

You can deploy Style Intelligence web services with native support for WebLogic 7. To do this, create a WAR file containing the description of SOAP services, supporting files, and the Style Intelligence JAR files. The structure for the WAR file should follow the directory structure below:

- *WEB-INF/*: *webservices.xml*
- *WEB-INF/classes*: *repository.xml*, *sree.properties*, *datasource.xml*, *query.xml*, and other xml files in the home directory
- *WEB-INF/lib*: *etools.jar*, *sree.jar/bisuite.jar*, *wsdl4j.jar*, and other JARs required for report execution

3.1.3 WebSphere

WebSphere 5.x uses Axis as its SOAP engine. Therefore you should follow the directions above for *Apache (Axis2)*.

3.2 Creating the SoapRepository Client

This section explains how to create a Java SOAP client for the *SoapRepository* service. The instructions below are appropriate for Metro 2.0, the default SOAP implementation.

Follow the steps below:

1. Make sure the *SoapRepository* service is deployed and running on the host server. For Metro, do this by entering the following URL:

   ```
   http://machine:port/webapp/SoapRepository
   (e.g., http://localhost:8080/sree/SoapRepository)
   ```

2. Create the client Java Stubs:

   a. Download and configure Metro version 1.5 (http://metro.java.net).

   b. Choose a folder where the Stub files will be created. (Preferably create a new one). We will refer to this folder as `{StubHome}`

   c. Create two folders in `{StubHome}`, e.g. '*classes*' and '*source*' or any other suitable names.

   d. Use the `{MetroHome}/bin/wsimport` utility to create the stubs with the following command:

   ```
   wsimport.bat -s {folder1} -Xendorsed {URL}?wsdl
   ```

For example, change the working directory to *{StubHome}/classes*, and then enter the following command:

```
{MetroHome}/bin/wsimport.sh -s ../source -Xendorsed
http://localhost:8080/sree/SoapRepository?wsdl
```

This will compile the Java stub files into the current directory and optionally save the source code in the 'source' folder.

Later, when you compile the client, the only resource you need on the classpath is this stub directory. For example:

```
set CLASSPATH={StubHome}\classes;.
```

3.2.1 Example: Executing a Report

A commonly utilized service is report generation. You can save the generated report locally on the client machine in an exported format (PDF, EXCEL, RTF etc).

In the following example you will generate a report called 'Ad Hoc', and then save the report as a PDF file (*AdHoc.pdf*) on the client machine. The 'Ad Hoc' report is deployed in the 'Tutorial' folder of the repository, and its accepts a parameter called '`state`', for which you will be prompted at the command line. Follow the steps below:

1. Save the client below (Listing 1. *MySoapClient.java Source Code*) as '*MySoapClient.java*'.

2. Add the stub classes to your classpath (see *Creating the SoapRepository Client*) and compile the client. For example:

   ```
   set CLASSPATH={StubHome}\classes;.
   javac MySoapClient.java
   ```

3. Run the client:

   ```
   java MySoapClient
   ```

4. Input a value (e.g., 'NJ') for the state parameter when the prompt appears at the command line.

The client calls the appropriate stub classes to generate and export a PDF version of the report (*AdHoc.pdf*) in the specified location.

Listing 1. MySoapClient.java Source Code

```
Note: Add the same parameter name multiple times to pass an array
of values.
```

```
import inetsoft.sree.soap.*;
import java.io.BufferedReader;
import java.io.FileOutputStream;
```

```
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.*;

public class MySoapClient {
  public static final String THE_REPORT = "Tutorial/Ad Hoc";
  public static final String OUTPUT_FILE = "./AdHoc.pdf";

  public static void main(String[] args) {
    final SoapRepositoryService service = new
SoapRepositoryService();
    final SoapRepository repository =
service.getSoapRepositoryPort();

    //Login with user, password and locale
    String ticket = null;

    try {
      ticket = repository.login("admin", "admin", null);
    }
    catch(Exception e) {
      e.printStackTrace();
      return;
    }

    //List replets
    List<RepositoryEntryStruct> replets =
repository.getReplets(ticket, null);

    for(RepositoryEntryStruct entry : replets) {
      System.out.println(entry.getPath());
    }

    //Get replet Parameters, if any
    RepletParametersStruct rps =
repository.getRepletParameters(ticket, THE_REPORT);
    RepletRequestStruct rrs =
readParameters(rps.getParameterNames());

    //Execute the report
    final String repletID = repository.executeReplet(ticket,
THE_REPORT, RepletType.REPLET, rrs);

    //Export the report
    final String exportID = repository.export(ticket, repletID,
FormatType.PDF);

    FileOutputStream output = null;
    try {
      output = new FileOutputStream(OUTPUT_FILE);
      byte[] buf = null;
      while((buf = repository.nextBlock(ticket, exportID)) != null)
{
        output.write(buf);
      }

      output.flush();
    }
    catch(IOException ioe) {
      ioe.printStackTrace();
    }
    finally {
      try {
        output.close();
      }
      catch(Exception e) {}
    }

    //Destroy the report when finished
    repository.destroyReplet(ticket, repletID);
    repository.logout(ticket);
  }

  /**
  *
  * @param parameterNames List of replet's parameters.
  * @return RepletRequestStruct with parameter values input by
user.
  */
  public static RepletRequestStruct readParameters(final
List<String> parameterNames) {
```

```
    //Create a replet request, to hold parameters
    final RepletRequestStruct rrs = new RepletRequestStruct();
    rrs.setName("create");

    List<String> reqParamNameList = rrs.getParamNames();
    List<ParameterValue> reqParamValueList = rrs.getParamValues();

    for(int i = 0; i < parameterNames.size(); i++) {
      System.out.println("Parameter Name: " +
parameterNames.get(i));
      BufferedReader buf = null;
      try {
        //Read in parameter values from command line.
        //You can also set them programatically.
        System.out.print("Input the value:");
        buf = new BufferedReader(new InputStreamReader(System.in));
        String value = buf.readLine();

        //Add the parameters to the RepletRequestStruct
        reqParamNameList.add(parameterNames.get(i));

        //Create a ParameterValue object to hold a string parameter
        ParameterValue pvalue = new ParameterValue();
        pvalue.setType(ParameterValueType.STRING);
        pvalue.setValue(value);

        reqParamValueList.add(pvalue);
      }
      catch(Exception e) {
        e.printStackTrace();
      }
      finally {
        try {
          buf.close();
        catch(Exception e) {}
      }
    }

    return rrs;
  }
}
```

3.2.2 Example: Emailing a Report

The client may additionally take advantage of server-side functions such as emailing a report:

```
//Email the report
repository.mailTo(ticket, repletID,
"recipient@inetsoft.com", "sender@inetsoft.com", "subject",
"message text", FormatType.PDF);
```

3.2.3 Example: Archiving a Report

The following example demonstrates how to save a report to the archive:

```
//Save in archive

//Define permission structure
PermissionStruct ps = new PermissionStruct();
List  readUserList = ps.getReadUsers();
readUserList.add("admin");

//Define archive rule
ArchiveRuleStruct ar = new ArchiveRuleStruct();
ar.setType(ArchiveRuleType.AGE);
ar.setDay(3);

//Save the report in the archive
```

```
repository.saveInArchive(ticket, repletID, "archived
customer list", FormatType.PDF, ps, ar, "comments");
```

3.2.4 Example: Scheduling a Server Task

The following example demonstrates how to schedule a task on the server:

```
//Create the scheduled task
ScheduleTaskStruct task = new ScheduleTaskStruct();
task.setName("Task Name");
task.setOwner("admin");
task.setEnabled(true);

//Set time condition
TimeConditionStruct condition = new TimeConditionStruct();
condition.setType(TimeConditionType.AT);
XMLGregorianCalendar cal =
DatatypeFactory.newInstance().newXMLGregorianCalendar();
cal.setYear(2009);cal.setMonth(12);cal.setDay(14);
cal.setTime(12,0,0);
condition.setDate(cal);

//Set replet action
RepletActionStruct action = new RepletActionStruct();
//executes the Master Detail report
action.setRepletName("Master Detail");
//schedule the report to be sent to foo@inetsoft.com
action.setEmails("jamshedd@inetsoft.com");
action.setFormat(FormatType.PDF);
action.setMessage("this is the report we talked about.");

List taskRepletActionList  = task.getRepletActions();
List taskTimeConditionList = task.getTimeConditions();
taskRepletActionList.add(action);
taskTimeConditionList.add(condition);

//Save the scheduled task
repository.setScheduleTask(ticket, "Task Name", task);
```

3.3 Creating a DataService Client

This section explains how to create a Java SOAP client for the *SoapDataService* service. This is a service which is used to execute queries and data worksheets. The instructions below are appropriate for Metro 2.0, the default SOAP implementation.

Follow the steps below:

1. Make sure the ‘SoapRepository’ service is deployed and running on the host server. For Metro, do this by entering the following URL: *http://machine:port/webapp/SoapDataService* (e.g., http://localhost:8080/sree/SoapDataService).

2. Create the client Java Stubs:

   a. Download Metro (https://metro.dev.java.net).

b. Choose a folder where the stub files will be created. (Preferably create a new one). We will refer to this folder as `{StubHome}`.

c. Create two folders in `{StubHome}`, e.g. ‘classes’ and ‘source’ or any other suitable names.

d. Use the `{MetroHome}/bin/wsimport` utility to create the stubs with the following command:

```
wsimport.bat -s {folder1} -d {folder2} -keep -Xendorsed
{URL}?wsdl
```

For example, change the working directory to `{StubHome}`, and enter the following command:

```
set PATH=%METRO_HOME%\bin;%PATH%
wsimport -s source -d classes -keep -Xendorsed http://
localhost:8080/sree/SoapDataService?wsdl
```

This will create the Java stub files in the classes folder and source code (optional) in the source folder.

Sample client code for the ‘SoapDataService’ service is shown below. Make sure the stub classes (described above) are added to your classpath.

```
import java.io.FileOutputStream;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.util.List;
import java.util.Calendar;
import javax.xml.datatype.XMLGregorianCalendar;
import javax.xml.datatype.DatatypeFactory;
import inetsoft.sree.dataserv.*;

public class MyDataClient {
  public static String TIME_INSTANT_FORMAT = "MM/dd/yyyy HH:mm:ss";
  public static String DATE_FORMAT = "MM/dd/yyyy";
  public MyDataClient(){}

  public static void main(String[] args) throws Exception {
    SoapDataServiceService service = new SoapDataServiceService();
    SoapDataService port = service.getSoapDataServicePort();

    //Login with user, password and locale
    String ticket = port.login("anonymous", "password");

    //List available assets
    List<String> paths = port.getPaths(ticket);
    System.out.println("PATHS: " + paths);

    //Aqcuire meta-data for 'orders by state' query
    XTableStruct table = port.getMetaData(ticket, "Orders:orders by
state");

    //List column types for 'orders by state' query
    List<String> colTypes = table.getColTypes();
    System.out.println("TYPES: " + colTypes);

    //Obtain query parameters, if any
    List<UserVariableStruct> variableNamesList =
port.getParameters(ticket, "Orders:orders by state");

    //Create a VariableTableStruct to hold any parameters
    VariableTableStruct vts = new VariableTableStruct();
    List<String> vtsParamNameList = vts.getVariableNames();
```

```
    List<ParameterValue> vtsParamValueList =
vts.getVariableValues();

    for(int i = 0; i < variableNamesList.size(); i++) {
      try {
        UserVariableStruct v =
(UserVariableStruct)variableNamesList.get(i);
        System.out.println("Parameter Name: " + v.getLabel());
        System.out.println("Parameter Type: " + v.getXType());
        System.out.println("Input the value
("+TIME_INSTANT_FORMAT+") :");
        BufferedReader buf = new BufferedReader(new
InputStreamReader(System.in));
        String value = buf.readLine();

        //add the parameters to the VariableTableStruct
        vtsParamNameList.add(v.getLabel());
        ParameterValue pvalue = new ParameterValue();
        pvalue.setValue(value);
        pvalue.setType(ParameterValueType.TIME_INSTANT);
        pvalue.setFormat(TIME_INSTANT_FORMAT);
        vtsParamValueList.add(pvalue);
      }
      catch(Exception e){
        System.err.println(e);
      }
    }

    String rid = port.execute(ticket, "Orders:orders by state",
vts);
    table = null;

    while((table = port.nextBlock(ticket, rid, 50)) != null) {
      //Print header rows
      for(int row = 0; row < table.getHeaderRowCount(); row++) {
        List<String> xrow =
table.getObjectRows().getObjectRow().get(row).getRow();
        System.out.println(xrow);
      }

      //Print body rows
      for(int row = table.getHeaderRowCount(); row <
table.getRowCount();row++) {
        List<String> xrow =
table.getObjectRows().getObjectRow().get(row).getRow();
        Object[] data = new Object[table.getColCount()];

        for(int col = 0; col < table.getColCount(); col++) {
          String colType = colTypes.get(col);
          String colValue = xrow.get(col);

          //Test return type
          if("java.lang.Boolean".equals(colType)) {
            data[col] = colValue == null || "".equals(colValue) ?
null : Boolean.valueOf(colValue);
          }
          else if("java.lang.Byte".equals(colType)) {
            data[col] = colValue == null || "".equals(colValue) ?
null : Byte.valueOf(colValue);
          }
          else if("java.lang.Character".equals(colType)) {
            data[col] = colValue == null || "".equals(colValue) ?
null : Character.valueOf(colValue.charAt(0));
          }
          else if("java.lang.Double".equals(colType)) {
            data[col] = colValue == null || "".equals(colValue) ?
null : Double.valueOf(colValue);
          }
          else if("java.lang.Float".equals(colType)) {
            data[col] = colValue == null || "".equals(colValue) ?
null : Float.valueOf(colValue);
          }
          else if("java.lang.Integer".equals(colType)) {
            data[col] = colValue == null || "".equals(colValue) ?
null : Integer.valueOf(colValue);
          }
          else if("java.lang.Long".equals(colType)) {
            data[col] = colValue == null || "".equals(colValue) ?
null : Long.valueOf(colValue);
          }
          else if("java.lang.Short".equals(colType)) {
```

```
            data[col] = colValue == null || "".equals(colValue) ?
null : Short.valueOf(colValue);
          }
          else if("java.util.Date".equals(colType)) {
            data[col] = colValue == null || "".equals(colValue) ?
null : new java.util.Date(Long.parseLong(colValue));
          }
          else if("java.sql.Date".equals(colType)) {
            data[col] = colValue == null || "".equals(colValue) ?
null : new java.sql.Date(Long.parseLong(colValue));
          }
          else {
            data[col] = colValue == null || "".equals(colValue) ?
null : colValue;
          }
        }

        System.out.print("[");
        for(int i = 0; i < data.length; i++) {
          if(i > 0) {
            System.out.print(", ");
          }

          System.out.print(String.valueOf(data[i]) + ":" + (data[i]
== null ? "null" : data[i].getClass().getName()));
        }
        System.out.println("]");
      }
    }

    port.logout(ticket);
  }
}
```

Compile the client in the usual manner. For example, if the client class is saved as *MyDataClient.java*, compile it as follows:

```
javac MyDataClient.java
```

To run the client, enter the following:

```
java MyDataClient
```

4 Design for Multi-Tenant Environment

In certain deployment scenarios, you may need to provide different groups of users (i.e., “tenants”) with access to different sets of data. This is called a *multi-tenant* environment. For example, if you are serving user groups from different departments or organizations, you may need to give each tenant access to data stored in a unique database or schema (different login permissions, etc.).

To facilitate design of a multi-tenant environment, Style Intelligence allows you to define independent data *connections* for each tenant. Each tenant can access only the unique connections for which they have privileges. Additionally, you can define a common set of data that is accessible to all tenants.

The steps below guide you through the process of configuring Style Intelligence for multi-tenancy.

1. Add database connections to your Data Source. Each connection may access a different database or schema that is specific to a particular tenant. See *Adding Additional Connections* in the *Advanced Data Modeling* for complete information.

2. Extend a common Physical View by incorporating database tables/columns from a tenant-specific connection. The extended physical view will provide the common tables/columns as well as tables/columns that are specific to the particular tenant. See *Extending a Physical View* in the *Advanced Data Modeling* for complete information.

3. Extend a common Logical Model by defining entities/attributes based on tables/columns from a tenant-specific connection (extended physical view). The extended logical model will provide the common entities/attributes as well as entities/attributes specific to the particular tenant. See *Extending a Logical Model* in the *Advanced Data Modeling* for complete information.

4. Assign the appropriate permissions for *connections* and *actions* to users and groups. See *Component/Object Permissions* in the *Advanced Server Administration* for complete information.

Note: *Report Pregeneration* and *Report Bursting* are not available in a multi-tenant environment.

5. Set appropriate aliases for reports, folders, Worksheets, and Viewsheets. You can use aliases to provide similar names to multiple tenants (e.g., a “Marketing” folder) while maintaining distinct names

for administration (e.g., “Marketing-Company A”, “Marketing-Company B”, etc.). See the related links for details on setting aliases.

See Also

Configuring Reports, in *Advanced Server Administration*, to set report aliases.

Creating Folders in *Advanced Server Administration*, to set folder aliases.

Setting Viewsheet Options, in *Advanced Dashboard Design*, to set Viewsheet aliases.

Setting Global Worksheet Options, in *Advanced Data Mashup*, to set Worksheet aliases.

5 Accessing the User Session with SRPrincipal

It is often desirable to set certain custom properties for a user session. While these properties are external to Style Intelligence, they can still be accessed from within the Style Intelligence environment through the SRPrincipal object that is associated with the user session.

The following sections explain how to access and modify the SRPrincipal object.

5.1 Accessing the SRPrincipal Object

The following sections explain the various methods of accessing the SRPrincipal object.

5.1.1 Accessing SRPrincipal in Custom JSP/Servlets

The SRPrincipal object can be accessed from the HttpSession object within a custom JSP or Servlet. You can add or set the properties of the SRPrincipal object using the *setProperty()* method.

```
<%@ page import="inetsoft.sree.security.*,
inetsoft.sree.RepletRepository" %>
<%
SRPrincipal p;
p = (SRPrincipal)
session.getAttribute(RepletRepository.PRINCIPAL_COOKIE);
if(p == null) {
   p = new SRPrincipal();
}

// setting a custom property
p.setProperty("property_name", "property_value");

// setting the locale using the locale string
p.setLocale(new Locale("en","US"));

session.setAttribute(RepletRepository.PRINCIPAL_COOKIE, p);
%>
```

5.1.2 Accessing SRPrincipal via Login Listener

You can modify the SRPrincipal object created after form-based sign-on to set user-specific properties, which you can later access in VPM script, etc. See *Accessing SRPrincipal in Script* and *HTTP Request, Session, and Principal* in *Report Scripting*.

To modify the SRPrincipal object when using form-based SSO, implement a *LoginListener* to be called at the time the user is authenticated (whether during live login or scheduled task). Follow the steps below:

1. Implement the `inetsoft.sree.security.LoginListener` interface's single method `userLogin(LoginEvent)`. Use `LoginEvent.getPrincipal()` to obtain the SRPrincipal object, and `SRPrincipal.setProperty()` to assign a custom property. For example:

```
import inetsoft.sree.security.*;

public class MyLoginListener implements LoginListener {
   public void userLogin(LoginEvent event) {
      SRPrincipal prin = event.getPrincipal();
      prin.setProperty("myprop", "myval");
   }
}
```

Note: To customize the "Welcome" text displayed in the default portal, call SRPrincipal.setAlias("Alternate Name").

2. In the `sree.propeties` file, set property `sree.security.listeners` to a comma-separated list of your fully qualified LoginListener class names. For example:

```
sree.security.listeners=MyLoginListener
```

See *Accessing SRPrincipal in Script* for information on accessing an SRPrincipal property (default or custom) in script.

See Also

Form-Based Single Sign-On, for instructions on configuring form-based SSO.

5.1.3 Accessing SRPrincipal in Script

You can access the SRPrincipal object within VPM trigger scripts and report scripts (in the onLoad and onInit handlers, or in element-level script).

```
var p = parameter['__principal__'];

// getting a custom property:
p.getProperty("property_name");

// Getting a common SRPrincipal property (user locale):
p.getLocale();
```

See Also

parameter.__principle__, in *Advanced Dashboard Scripting*, for information Viewsheet access.

Advanced Dashboard Scripting, for information on adding Viewsheet scripts.

Report Scripting, for information on adding report scripts.

Accessing SRPrincipal via Login Listener, for information on adding custom properties.

Trigger Scripts, in *Advanced Data Modeling*, for information on VPM scripts.

5.2 SRPrincipal Properties

This section described the key properties of the SRPrincipal object.

5.2.1 setProperty()

The `setProperty()` method inserts a property (string) into the SRPrincipal object. The sample below uses `setProperty()` within a Login Listener:

```
import inetsoft.sree.security.*;

public class MyLoginListener implements LoginListener {
  public void userLogin(LoginEvent event) {
    SRPrincipal prin = event.getPrincipal();
    prin.setProperty("myprop", "myval");
  }
}
```

The corresponding `getProperty()` method allows you to retrieve a property value.

```
prin.getProperty("myprop")
```

In a non-SSO setting, set the '`__internal__`' property when you create the SRPrincipal object, as shown below:

```
SRPrincipal prin = new SRPrincipal(user,...);
prin.setProperty("__internal__", "true");
```

5.2.2 setParameter()

The `setParameter()` method inserts a parameter and its value into the SRPrincipal object. The sample below uses `setParameter()` within an SSO request filter:

```
HttpServletRequest hrequest = (HttpServletRequest) request;
HttpSession session = hrequest.getSession();
SRPrincipal principal =
(SRPrincipal)session.getAttribute(RepletRepository.PRINCIPA
L_COOKIE);
Principal.setParameter("State", new String[] {"NJ", "NY",
"CT"})
```

A parameter value specified with `setParameter()` can be accessed in the following contexts:

- In reports and Viewsheets via the `parameter` script keyword. (See *parameter* in *Report Scripting* and *parameter* in *Advanced Dashboard Scripting*.)
- In a parameterized condition of a report data binding. (See *Parameterizing a Condition* in *Advanced Report Design*.)

- In a parameterized condition on a Worksheet Data Block. (See *Using a Parameter in a Condition* in *Advanced Data Mashup*.)
- In a parameterized condition on a Viewsheet component. (See *Filter Conditions* in *Advanced Dashboard Design*.)

When accessing the parameter in a condition (Viewsheet, Worksheet, or report), the name of the condition variable should exactly match the parameter name specified by `setParameter()`.

5.3 Use Case: Simulating User Sessions

The example presented here will provide administrative users the ability to simulate different user sessions without explicitly logging in as that user. To login as a user, see *Emulating a User Login* in *Advanced Server Administration*.

The administrator would login to the InetSoft server, and access a custom JSP page which sets custom properties on the SRPrincipal object. See *Accessing the SRPrincipal Object* for more details.

```
inetsoft.sree.security.SRPrincipal p =
(inetsoft.sree.security.SRPrincipal)session.getAttribute
(inetsoft.sree.RepletRepository.PRINCIPAL_COOKIE);

// setting a custom property
p.setProperty("sim_user", "david");
```

These custom properties can be accessed within the script of a VPM and used to set a user/role based filtering parameter defined in the ‘Conditions’ tab of the VPM.

```
VPM Lookup Trigger:

var p = parameter['__principal__'];

if(!isNull(p)) {
   parameter['usr'] = p.getProperty("sim_user");
}
```

Note that `parameter['usr']` would be a parameter defined in a clause of the VPM filtering conditions (in the **Conditions** tab), for example:

```
sales_employees.first_name [is] [equal to] $(usr)
```

Reserved parameters such as `$(_USER_)` or `$(_ROLES_)` cannot be modified (set) in VPM script and should not be used directly in VPM filtering conditions.

APPENDIX A: Changing the Name of the 'Sree' Web Application

If you change the Style Intelligence web application name to something other than "*sree*", you must modify the values **highlighted in bold** in the following files to reference the correct web application context root and ServletRepository servlet mapping:

Please stop your web application server before making these changes.

- File: \webapps\{*contextRoot*}\WEB-INF\config\services\ConfigService.properties

```
servlet.insecure=/Reports/dashboard
servlet.secure=/Reports/dashboard
```

- File: \webapps\{*contextRoot*}\WEB-INF\classes\sree.properties

```
replet.repository.servlet=/sree/Reports
dashboard.uri=/sree/Reports/dashboard
dashboard.context=/sree
repository.replet.servlet=/Reports
```

- File: \webapps\{*contextRoot*}\WEB-INF\web.xml

```
<servlet-mapping>
   <servlet-name>replets</servlet-name>
   <url-pattern>/Reports/dashboard/*</url-pattern>
</servlet-mapping>

(The url-pattern should match the dashboard.uri property)
```

APPENDIX B: Common Portal Frameworks

The following sections discuss issues related top some common server and portal frameworks.

B.1 IIS Web Server

Microsoft IIS (Internet Information Server) is a commonly used web server on the windows platform. From a request-serving perspective, IIS natively supports serving only static files but has the facility to execute add-ons/plug-ins. This applies to all IIS versions. These add-ons/plug-ins are configured as 'Application Mappings' on IIS and are mapped to certain resource extensions (e.g., "`.jsp`"). IIS simply executes the configured add-on to handle a particular resource extension whenever it is requested. IIS is not inherently J2EE compliant and therefore cannot directly deploy servlets/JSPs. Since Style Intelligence is a 100% Java tool, it can only be deployed on a J2EE compliant application server. There are many available options for Java add-ons/plug-ins, and a few of those options are discussed here.

Discrete Web Server and Reporting Server (Recommended Approach)

Since InetSoft Enterprise products are servlet based, all commands to the reporting engine are sent via a servlet (repository servlet) in the form of HTTP URL requests with appropriate parameters. It is a common practice to run your reporting server separately and simply use HTTP URLs to link to the report server. For example, your portal framework could contain simple HTML pages which have framesets pointing to the report server URL as their source. See *Integrating the Web User Interface*.

IIS Tomcat Redirector

A popular add-on which allows IIS to run JSPs/Servlets on Tomcat is `isapi_redirect`. This plug-in allows IIS to redirect requests for JSPs and servlets to a Tomcat server. It still requires you to run Tomcat's web server separately on a certain port so that JSP/Servlets can be configured and executed. It registers an ISAPI on IIS to steal JSP requests from IIS request processing, opens a connection to the Tomcat web server, and shuttles these requests to Tomcat (i.e., it proxies the request). In this case, Style Intelligence will be deployed on a Tomcat server but will receive requests from an IIS server.

B.2 SharePoint Portal Framework

SharePoint is a popular portal framework developed by Microsoft (usually deployed on an IIS server). The 'Contents' of a Share Point portal consists of one or more components referred to as 'Web Parts'. A Web Part may retrieve its content from any third party source by simply specifying the appropriate URL. Therefore, by adding a new 'Web Part' and specifying

the appropriate content URL we can display an InetSoft dashboard or even a single replet as a 'Web Part', e.g., `http://localhost:8080/sree/Reports?op=portal_portalWelcome`. See *Integrating the Web User Interface* and Appendix C: *Servlet Repository Parameters*.

Seamless integration is only possible when the Style Intelligence Report Server is running on the same host machine as the SharePoint Portal. If the Report Server is running on another host machine, Style Intelligence replets and dashboards will be accessible, however, browser security restrictions (due to cross-domain references within the same browser frame) will disable interactive operations. To enable full functionality, use the Style Intelligence Proxy Servlet, a lightweight component which runs on the same machine as your portal, and proxies all requests to the reporting server. For information on how to set up the Style Intelligence Proxy Servlet, see the *Advanced Server Administration*.

APPENDIX C: Servlet Repository Parameters

The Servlet Repository generates DHTML output according to the parameters in HTTP requests. The parameters are embedded in the generated DHTML and are normally not used directly by an application to access the servlet.

This *Appendix* contains a partial list of the parameters for the servlet. You can use these parameters to construct a URL to access the server from application client-side code. The HTTP request can be submitted with both the GET and POST methods.

Also see *Integrating the Web User Interface* for information on integrating report components using the "op" codes described below.

C.1 Displaying a Viewsheet

Use the following syntax to open a Viewsheet via direct URL:

```
{servlet URL}?op=vs&path={viewsheet path}&edit=true|false

e.g.,

http://localhost:8080/sree/Reports?op=vs&path=/Explore
```

To display the Viewsheet using HTML5 rather than Flash, use "`op=vs_html`". For example:

```
http://localhost:8080/sree/Reports?op=vs_html&path=/Explore
```

Note that Viewsheets displayed using HTML5 do not support Ad Hoc Editing features for charts and crosstabs.

C.2 Passing a Parameter to a Viewsheet

`Note: to pass a value to a parameterized datasource, prefix the parameter name with "__HYPERLINK_".`

To directly pass parameters to a Viewsheet via the URL, prefix the parameter name with "`VSPARAM_`". For example, if the parameter name is `param1`:

```
http://localhost:8080/sree/Reports?op=vs&path=/
Explore&VSPARAM_param1=val1
```

The transmitted parameter will be available in the Viewsheet through the `parameter` script object and in parameterized filtering conditions.

See Also

parameter, in *Advanced Dashboard Scripting*, for information about accessing parameters.

Filter Conditions, in *Advanced Dashboard Design*, for information on parametrizing filters.

C.3 Displaying a Report

Use the following syntax to open a report via direct URL:

```
{servlet URL}?op=frameReplet&name={replet name}

Example:

http://localhost:8080/sree/
Reports?op=frameReplet&name=OrderList
```

To suppress the report's parameter sheet, add the property "`disableParameterSheet=true`" to the URL.

```
Example:

http://localhost:8080/sree/
Reports?op=frameReplet&name=OrderList&disableParameterSheet
=true
```

C.4 Displaying a Report Chart as an Image

Use the following syntax to display a chart from a specified report via direct URL. The chart is rendered as a PNG image.

```
{servlet URL}?op=frameReplet&name={replet
name}&outtype=PNG&element_id={GraphElementID}

e.g.,

http://localhost:8080/sree/
Reports?op=frameReplet&name=MyReport&outtype=PNG&element_id
=Graph1
```

C.5 Passing a Parameter to a Report

To directly pass parameters to a report via the URL, prefix each parameter name with "`&`". For example, if the parameter name is `param1`:

```
{servlet URL}?op=frameReplet&name={replet name}&param1=val1

e.g.,

http://localhost:8080/sree/
Reports?op=frameReplet&name=OrderList&customer=Rutgers+Bank
&salesperson=Eric
```

`Note: to pass a value to a parameterized datasource, prefix the parameter name with "__HYPERLINK_".`

The transmitted parameter will be available in the report through the `parameter` script object and in parameterized filtering conditions.

See Also

parameter in *Report Scripting*, for information about accessing parameters.
Parameterizing a Condition in *Advanced Report Design*, for information on parametrizing filters.

C.6 Exporting a Report

Use the following syntax to export a report to a particular format via direct URL:

```
{servletURL}?op=frameReplet&name={replet name}&outtype={exp
ort_type}&param1=val1...

e.g.,

http://localhost:8080/sree/
Reports?op=frameReplet&name=OrderList&outtype=PDF&customer=
Rutgers+Bank&salesperson=Eric
```

The `{export_type}` can be one of the following:

- `PDF`: PDF file
- `EXCEL`: Excel spreadsheet
- `EXCEL_SHEET`: Excel spreadsheet without pagination
- `RTF`: RTF document
- `RTF_LAYOUT`: RTF document with pagination
- `HTML`: HTML document
- `HTML_BUNDLE`: HTML document and images in a zip file,
- `CSV`: Comma separated file.

C.7 Displaying a Page of a Generated Report

Use the following syntax to open a particular report page via direct URL:

Note: Page numbering starts with 0.

```
{servlet URL}?op=Page&ID={replet ID}&pn={page num}&mode=0

e.g.,

http://localhost:8080/sree/Reports?op=Page&ID={replet
ID}&pn=0&mode=0
```

C.8 Displaying the Repository Tree

Use the following syntax to open the repository listing via direct URL:

```
{servlet URL}?op=EmbeddedTree

e.g.,

http://localhost:8080/sree/Reports?op=EmbeddedTree
```

C.9 Displaying the Dashboard Page

Use the following syntax to open the dashboard page via direct URL:

```
{servlet URL}?op=portal_dashboard

e.g.,

http://localhost:8080/sree/Reports?op=portal_dashboard
```

C.10 Displaying the Dashboard Configuration Page

Use the following syntax to open the dashboard configuration page via direct URL:

```
{servlet URL}?op=dashboard_configuration

e.g.,

http://localhost:8080/sree/
Reports?op=dashboard_configuration
```

C.11 Displaying an Ad Hoc Wizard

Use the following syntax to open a particular Ad Hoc Wizard via direct URL:

```
{servlet URL}?op=adhocwizard&action=frame&wizard={Wizard
Name}

e.g.,

http://localhost:8080/sree/
Reports?op=adhocwizard&action=frame&wizard=Table+Wizard
```

C.12 Displaying the Scheduler Task Page

Use the following syntax to open the Scheduler Task page via direct URL:

```
{servlet URL}?op=scheduler_tasks

e.g.,

http://localhost:8080/sree/Reports?op=scheduler_tasks
```

C.13 Passing Credentials and Locale to Report Server

Use the following syntax to pass login credentials and locale to the report server via direct URL:

```
{Server
URL}?userid={username}&password={passwd}&locales_list={loca
le}

e.g.,

http://localhost:8080/sree/
Reports?userid=admin&password=admin&locales_list=fr_CA
```

See Also

Localization/Internationalization, in *Advanced Server Administration*, for details on localizing report and interface elements.

Configuring Single Sign-On (SSO), for best practices in configuring a single sign-on application architecture.

C.14 Pinging the Report Server

Use the following syntax to test the connection status of the Report Server via direct URL:

```
{servlet URL}?op=ping

e.g.,

http://localhost:8080/sree/Reports?op=ping
```

A response of 'OK' indicates that the Report Server is receiving connections.

APPENDIX D: Configuring Style Intelligence for JSP Tags

The following sections describe an approach to integration that relies on the JSP tag library for a J2EE environment. In general, this is not as flexible as the approach based on IFrames described in *Integrating the Web User Interface*. Some of the major differences between the approaches are as follows:

`Use the IFrame approach whenever possible.`

- The JSP tag approach can only be used to embed reports, not Viewsheets.
- The JSP tag approach provides no ad hoc editing capabilities.
- The JSP tag approach requires greater technical expertise.
- The JSP tag approach permits greater control over placement of components on the page.

To start using the Style Intelligence JSP tags, you must place *bisuite.jar* and *etools.jar* in the *WEB-INF/lib* directory of your web application. Include the tag library definition in your web application descriptor, *web.xml*, which is located in the *WEB-INF* directory of your web application. Simply add the following XML content to the file:

```
<taglib>
  <taglib-uri>sree.tld</taglib-uri>
  <taglib-location>/WEB-INF/lib/bisuite.jar
  </taglib-location>
</taglib>
```

For Style Report EE, specify *sree.jar* in place of *bisuite.jar* for all occurrences.

If your repository directory is a directory other than the `WEB-INF/classes` directory, you will need to include an initialization parameter as well:

```
<context-param>
  <param-name>sree.home</param-name>
  <param-value>/path/to/your/repository/directory</param-
value>
</context-param>
```

The value of this parameter is the same as the one you would use to set up the Servlet Repository.

In certain application servers, it is impossible to serve binary data through a JSP. The resources will fail to load and an exception will appear. To avoid this, a servlet, `inetsoft.sree.web.jsp.ResourceServlet`, is provided to serve binary resources. If you encounter this problem, register the servlet and specify the servlet's URI in the resourceUri attribute of the sree:create tag.

No additional configuration is needed in your web application. You are now ready to start writing JSP pages.

D.1 Tag Library Import

In order to use Style Intelligence tags in a JSP page, you need to import the tag library. This is done by using a JSP taglib directive:

```
<%@taglib uri="sree.tld" prefix="sree"%>
```

This tells the JSP that all tags that start with "<sree:" refer to the tag library defined in the web.xml file with the URI of sree.tld. The prefix is an XML namespace prefix. This means that any tag included in the page needs to be prefixed with "sree:". This prevents naming collisions if another tag library has a tag of the same name.

D.2 Cache Tag

The first tag that must be included in the JSP is the "`cache`" tag. Use the `cache` tag when you have multiple reports on the same JSP page, and want to refresh one report without regenerating the others (i.e., the others should be displayed from the cache). The `cache` tag has a single required attribute, `jspId`, which needs to be unique for the whole web application. This means that no JSPs may have a `cache` tag with the same `jspId`. There is one cache tag for the entire JSP page.

```
<sree:cache jspId="reportJSP"/>
```

D.3 Create Tag

The next tag needs to be the "`create`" tag. This tag specifies the report that will be embedded in the JSP. It also handles the generation of the report and serves all of the resources required by the report.

There cannot be any HTML content or JSP tags that generate content before this tag. The `create` tag handles serving images and static files such as cascading style sheets and JavaScript files. If there is any content before this tag, it will be included in the resource file and will corrupt it. If you require that there be content before the `create` tag, you must specify the resourceUri attribute. This attribute should be set to the URL of a Servlet Repository deployed in the same web application. The Servlet Repository and the JSP tag library share the same resources and report server, so there is very little additional overhead.

In WebLogic, there also cannot be any characters (i.e., new lines) outside of JSP tags before the create tag. WebLogic will print these characters to the output stream, corrupting the image file. Setting the resourceUri attribute will solve this problem.

The create tag has two required attributes, reportId and name. The reportId attribute is a unique string that identifies all subsequent tags as referring to this report. This allows you to include more than one report in a single JSP page. The reportId need only be unique within the JSP, not within the entire web application. The name attribute is the name of the replet as it is registered in the Replet Repository. The name should include all folders in its path, separated by a forward slash (/).

The create tag also has one optional parameter, processPage. This attribute determines the behavior of the JSP if the report is not yet available. If processPage is set to false, the remainder of the JSP is not processed and the text “Loading Report...” is displayed. If processPage is set to true, the JSP is processed and the content included in the body of the subsequent report tags is displayed in place of the report element.

```
<sree:create reportId="myReport" name="My Folder/My Report"
             processPage="true"/>
```

D.4 Parameter Tag

The create tag can also contain parameter tags. These tags pass report parameters to the report. The parameter tag has three required attributes: name, value, and type. The name attribute specifies the name of the parameter, the value attribute specifies the value of the parameter, and the type attribute specifies the data type of the parameter value. The allowed parameter types are listed in Appendix E: *JSP Tag Library Reference*. If the type is “date”, you also need to include the format attribute. The format attribute specifies the format in which the value is being passed and should use the format specified in java.text.SimpleDateFormat.

```
<sree:create reportId="report1" name="A Report"
processPage="false">
<sree:parameter name="State" value="NY" type="string"/>
<sree:parameter name="Order Date" value="2003-05-01"
   type="date" format="yyyy-MM-dd"/>
</sree:create>
```

If a report parameter has the same name as a JSP request parameter, it will be passed to the report without having to include a parameter tag. Using a parameter tag in this case will override the value of the JSP parameter.

D.5 Header Tag

The Header Tag must be present to ensure correct generation of the report.

In the HTML head tag, you need to include the report header tag. The only attribute passed to this tag is the reportId. The header tag is responsible for writing java script functions which are used by the report toolbar buttons (mail, export etc). It also writes some external CSS data for the presentation of the report.

```
<html>
   <head>
```

```
    <title>My Report Page</title>
    <sree:header reportId="myReport"/>
</head>
```

D.6 Toolbar and Body Tags

The Body Tag must be present to ensure correct generation of the report.

The HTML body tag must have the onLoad attribute defined with the scriptlet `<%=pageContext.getAttribute("sreeOnLoad")%>` appended to any onLoad java script you have defined. This is used for invoking java script written by the header tag when the page loads.

The toolbar and body can be placed anywhere within the HTML body. Both tags require the reportId attribute.

```
<body onLoad="initMyPage();<%=pageContext.getAttribute
("sreeOnLoad")%>">
   <p>This report shows sales for the state of New York.</p>

   <sree:toolbar reportId="myReport"/>
   <sree:body reportId="myReport">
     <p>Please wait while the report is being generated.</p>
   </sree:body>
```

The content enclosed in the body tag is written if the report is not yet available and the processPage attribute of the create tag is set to true. The toolbar tag can also optionally include content that will be written if the page is not available.

D.7 Button Tag

The button tag provides you with the means to fully customize any toolbar button on an individual report basis. It requires the reportId attribute as well as the type attribute. The type attribute specifies which toolbar button the tag inserts. The different types of tags are listed in Appendix E: *JSP Tag Library Reference*.

Additional attributes of the button tag include image, text, useImage and actionClass. These attributes allow you to change how each button is presented. The image attribute specifies the URL of the icon to use if the button is an image type button. The supported URL protocols and expected format of this attribute are defined in Appendix E: *JSP Tag Library Reference*. The text attribute specifies the text that should be used for the link if the button is a text type button. The useImage attribute specifies whether the button will use an image or text. The actionClass attribute specifies the fully qualified class name of the action class that is called when the button is clicked. This attribute is required when the button type is “user”. The values of these attributes will override the settings in the SREE properties file. If an attribute is omitted, the default setting will be used.

The button tag will only write the button if it would normally be displayed in the toolbar. For example, the previous page button would not be written on the first page, and if the user does not have analytic capabilities then the 'Ad Hoc' button will not be shown.

Like the toolbar and body tags, the button tag can contain content that will be written if the report is not yet available and can be placed anywhere in the HTML body.

```
<sree:button reportId="type="next-page"
   image="images/next.gif" useImage="true"/>
```

Note: It is often desirable to have an enabled and a disabled version for a button. For example, the 'next' button should be disabled on the last page of the report. This can be achieved by adding another file with the same name as the image file suffixed with '_d'. The two files should be in the same location, e.g., '*next.gif*' and '*next_d.gif*'

This is an example of a simple but complete JSP page with an embedded report.

```
Note: Functions such as Ad Hoc editing and Report Explorer are
unavailable when the reports are embedded in a JSP.
```

Listing 2. Embedding Report in a JSP

```
<%@taglib uri="sree.tld" prefix="sree"%>
<% String orderDate = request.getParameter("order_date"); %>
<sree:cache jspId="reportJsp"/>
<sree:create reportId="myReport" name="My Folder/My Report"
             processPage="true">
   <sree:parameter name="State" value="NY" type="string"/>
   <sree:parameter name="OrderDate"
      value="<%= orderDate %>" type="date"
      format="yyyy-MM-dd"/>
</sree:create>

<html>
   <head>
      <title>NY Sales Report</title>
      <sree:header reportId="myReport"/>
      <link rel="stylesheet" href="hostStyle.css"
          type="text/css"/>
   </head>
   <body onLoad="
      <%= pageContext.getAttribute("sreeOnLoad") %>">
      <h1>Sales Summary for NY on <%= orderDate %></h1>
      <hr>
      <sree:toolbar reportId="myReport"/>
      <sree:body reportId="myReport">
          <p>Please wait while the report is loading.</p>
      </sree:body>
   </body>
</html>
```

Listing 3. Using Individual Buttons in a JSP

```
<%@taglib uri="sree.tld" prefix="sree"%>
```

```
<sree:cache jspId="reportJsp"/>
<sree:create reportId="myReport" name="My Folder/My Report"
             processPage="false"/>
<html>
   <head>
      <title>Sales Report</title>
      <sree:header reportId="myReport"/>
   </head>
   <body onLoad="
      <%= pageContext.getAttribute("sreeOnLoad") %>">
      <sree:button reportId="myReport" type="first-page"/>
      <sree:button reportId="myReport" type="previous-page"
          image="http://myhost.com/image.gif"/>
      <sree:button reportId="myReport" type="page-box"/>
      <sree:button reportId="myReport" type="next-page"
         image="images/image.gif"/>
      <sree:button reportId="myReport" type="last-page"
         image="jar:/usr/java/pkg.jar!/images/image.gif"/>
      <sree:button reportId="myReport" type="pdf"
         image="resource:/images/image.gif"/>
      <sree:button reportId="myReport" type="export"
         image="file:///usr/share/images/image.gif"/>
      <sree:button reportId="myReport" type="reload"
         useImage="false" text="Reload Report"/>
      <sree:button reportId="myReport" type="user"
         actionClass="com.mycompany.report.ReportAction"/>
      <sree:body reportId="myReport"/>
   </body>
</html>
```

D.8 Parameters Containing Unicode Characters

If a report is passing a parameter with a unicode value, it must be decoded to be displayed properly. Use the provided JavaScript function '`byteDecode`' for this purpose, as shown below:

```
<script language="JavaScript" src="/inetsoft/sree/internal/
markup/encodeUtil.js"></script>
<script language="JavaScript" >
   var decoded = byteDecode(encoded_parameter);
</script>
```

APPENDIX E: JSP Tag Library Reference

The following appendices describe various JSP tags.

E.1 Cache Tag

The *cache* tag caches the content of the report page. This tag must be the first report tag in the page.

The *cache* tag is required.

ATTRIBUTE NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES
jspId	Unique ID of the cache	Yes	Any globally unique string.

E.2 Create Tag

The *create* tag generates the report and serves any resources required to display the report. This tag must be inserted after the *cache* tag, but before any other report tags. If there is any HTML content or JSP tags that write HTML content before the *create* tag, the *resourceUri* attribute must be set to the URL of the Servlet Repository.

The *create* tag is required. It can contain *parameter* tags.

ATTRIBUTE NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES
reportId	Unique ID of the report	Yes	Any string that is unique within the JSP
name	The name of the report	Yes	The name of any report in the report repository
processPage	Specifies if the JSP will be processed when the report is not available	No	true/false
resourceUri	The URI of the servlet or JSP that should be used to load report resources	No	Any valid URL
linkUri	The URI that should be used as the base of all links inside the report	No	Any valid URL

E.3 Parameter Tag

The *parameter* tag defines parameters that are passed to the report. This tag must be enclosed within a create tag.

The *parameter* tag is optional.

ATTRIBUTE NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES
name	The name of the parameter	Yes	The name of any parameter defined in the report
value	The value of the parameter	Yes	Any valid value
type	The data type of the parameter	Yes	string, number, date, boolean
format	The format of the value, if it is a date parameter	Only for dates	A format string as specified by java. text.SimpleDateFormat

E.4 Header Tag

The *header* tag includes the CSS style information, JavaScript and any imports required by the report. This tag must be placed within the HTML head tag.

The *header* tag is required.

ATTRIBUTE NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES
reportId	Unique ID of the report	Yes	The report ID specified in the create tag

E.5 Body Tag

The *body* tag writes the report page in the JSP. This tag may be placed anywhere in the HTML body.

The *body* tag is required. It can contain valid HTML or JSP content. This content will be written if the report is not available and the *processPage* attribute of the *create* tag is true.

ATTRIBUTE NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES
reportId	Unique ID of the report	Yes	The report ID specified in the create tag

E.6 Toolbar Tag

The *toolbar* tag inserts the report toolbar as displayed by the Servlet Repository. This tag may be placed anywhere in the HTML body.

The *toolbar* tag is optional. It can contain valid HTML or JSP content. This content will be written if the report is not available and the *processPage* attribute of the *create* tag is true.

ATTRIBUTE NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES
reportId	Unique ID of the report	Yes	The report ID specified in the create tag

E.7 Button Tag

The *button* tag inserts an individual toolbar button into the JSP. This tag may be placed anywhere in the HTML body. Any attributes that are not set will use the default value and all attributes that are specified will override the default settings in the SREE properties file.

The *button* tag is optional. The *button* tag may contain valid HTML or JSP content. This content will be written if the report is not available and the *processPage* attribute of the *create* tag is true.

ATTRIBUTE NAME	DESCRIPTION	REQUIRED	ALLOWED VALUES
reportId	Unique ID of the report	Yes	The report ID specified in the create tag
type	Specifies which button to insert	Yes	first-page, previous-page, page-box, next-page, last-page, find, find-next, PDF, refresh, archive, export, mail, print, customize, toc, menu, user

image	The URL of the image to use as the button's icon	No	Any valid URL. URLs beginning with resource:// will be loaded from the classpath of the web application. file:// URLs will be loaded from the file system of the server. URLs with the JAR: protocol will be loaded from a JAR file in the file system of the server. Any other URLs will be interpreted as HTTP URLs and will be used exactly as specified.
text	The text to use for a text-type button	No	Any string
useImage	Determines if the button will be a text or image type	No	true/false
actionClass	The fully qualified class name of the action that will be invoked when the button is clicked	Only for user-defined buttons	The name of any Java class that is on the web application's classpath

Common Reference

The following appendices discuss general functions that are useful across multiple Style Intelligence products:

- *General JavaScript Functions*

  Describes functions for processing and comparing text, dates, and numbers, and discusses iteration and flow control structures available in the JavaScript language.

- *Chart Script Tutorial*

  Describes scripting techniques for creating charts, specifying sub-series, setting axis information, decoration, and annotation.

- *Chart Script Reference*

  Reference to objects, properties, and methods available for scripting charts.

- *Using Style Studio*

  Describes various configuration options in Style Studio.

See Also

Style Intelligence Object Reference, in *Report Scripting* for functions specific to the reporting environment.

Advanced Dashboard Scripting, for functions specific to the Viewsheet environment.

APPENDIX JS: General JavaScript Functions

This section offers a brief introduction to common JavaScript language constructs and typical usage. It is not a reference document for JavaScript. For more detailed information on JavaScript, please consult the ECMA-262 specification or a JavaScript book. You can also find information on JavaScript syntax and constructs on the Mozilla web site.

JavaScript is an object-oriented programming language for performing computations and manipulating computational objects within a host environment. An *object* is a collection of properties and functions. A property can be a primitive type, a host object type, or a Java object.

The primitive types are standard in every JavaScript runtime and are independent of the host environment. These primitive object types include the following:

- *Global Object Functions*
- *String Object Functions*
- *Number Object Functions*
- *Date Object Functions*
- *Array Object Functions*
- *Math Object Functions*
- *Regular Expression Object Functions*

In addition to the primitive types, this Appendix describes a number of other functions and JavaScript constructs. Some of these functions require knowledge of JavaScript arrays. See *Array Object Functions* for information on how to use arrays. Columns within a table or Data Worksheet can also be treated as arrays.

See Also

Style Intelligence Object Reference, in *Report Scripting*, for more objects in a reporting environment.

Accessing Table Cells in Script, in *Advanced Data Mashup*, for information on scripting for Worksheets.

Advanced Dashboard Scripting, for information on scripting for Viewsheets.

JS.1 Global Object Functions

The Global object contains the properties and functions available for the entire scripting environment. The values and functions can be accessed anywhere as top-level properties and functions without any qualifier.

```
// convert a text field to an integer
var val = parseInt(TextField1.text);

// assign the half of the value to another text element
Text2.text = (val * 1 / 2).toString();
```

FORMATDATE(DATE, STRING)

Formats a date according to a simple date format and returns the result as a string.

Parameters

date	a java or javascript date object
format string	The format follows the Java SimpleDateFormat specification

Example

```
formatDate(new Date(), "hh 'o''clock' a, zzzz");
```

The date format uses the same syntax as in java.text.SimpleDateFormat.

A format string can be constructed using the characters listed in the table below.

SYMBOL	MEANING	PRESENTATION	EXAMPLE
G	era designator	Text	AD
y	year	Number	1996
M	month in year	Text & Number	July & 07
d	day in month	Number	10
h	hour in am/pm (1~12)	Number	12
H	hour in day (0~23)	Number	0
m	minute in hour	Number	30
s	second in minute	Number	55
S	millisecond	Number	978
E	day in week	Text	Tuesday
D	day in year	Number	189
F	day of week in month	Number	2 (2nd Web in July)
w	week in year	Number	27
W	week in month	Number	2
a	am/pm marker	Text	PM
k	hour in day (1~24)	Number	24
K	hour in am/pm (0~11)	Number	0
z	time zone	Text	Pacific Standard Time
‘	escape for text	Delimiter	
“	single quote	Literal	‘

Pattern letters can be repeated to obtain the desired date format. For the symbols marked as ‘Text’ for its presentation, if the number of pattern letters is 4 or more, the fill form is used; otherwise a short or abbreviated form is used if available. For the symbols marked as ‘Number’ for its presentation, the number of pattern letters is the minimum number of digits, and shorter numbers are zero-padded to this amount.

Some examples are shown below:

DATE AND TIME PATTERN	RESULT
“yyyy.MM.dd G ‘at’ HH:mm:ss z”	2006.08.07 AD at 02:19:35 PDT
“EEE, MMM d, ‘ ‘ yy”	Mon, Aug 7, ‘06
“h:mm a”	12:08 PM
“hh ‘o’ ‘clock’ a, zzzz”	12 o’clock PM, Pacific Daylight Time

FORMATNUMBER(NUMBER, STRING, STRING)

Formats a number according to the format string.

Parameters

```
number
format string       The format follows the Java
                    DecimalFormat specification
rounding string     Specifies how the number is to be
                     rounded
```

Example

```
formatNumber(3, '#,###.00','ROUND_HALF_UP');
```

A number format string can be constructed using the characters listed in the table below.

SYMBOL	MEANING
0	Digit
#	Digit, zero shows as absent
.	Decimal point
-	Minus sign
,	Grouping separator
E	Separates mantissa and exponent in scientific notation
;	Separates positive and negative sub-patterns
%	Multiply by 100 and show as percentage

UNDEFINED

This value is returned if a non-existent object or property is referenced. It is also the value of a variable before it is initialized.

EVAL(STRING)

Evaluate a string of JavaScript code without reference to a particular object. This function is useful for executing dynamically created JavaScript commands and expressions.

PARSEINT(STRING, RADIX)

Parse a string into integer. If the radix is omitted or undefined, it is assumed to be 10, unless the string begins with '0x' or '0X', in which case 16 is assumed.

PARSEFLOAT(STRING)

Parse a string into a decimal representation of a number.

PARSEDATE(STRING, FORMAT)

Parse a string into a date.

Parameters

string	string representation of the date
format	format string, or boolean value set to true for parsing a timeInstant object

Example

```
parseDate('2006-08-07', 'yyyy-MM-dd');
```

ENCODEURI(URI)

Produce a new URI with any special characters replaced by an appropriate encoding.

DECODEURI(URI)

Produce a new URI with escape sequences or UTF-8 encoding replaced by the characters they represent.

REGISTERPACKAGE(STRING)

Register the specified package to enable the use of all the classes in the package. All packages starting with `java`, `inetsoft`, `com`, or `org` are registered automatically. Packages may also be registered by setting the javascript.java.packages property for the application.

Example

```
registerPackage('inetsoft.report');
```

IMPORTPACKAGE(PACKAGE)

Enable the use of unqualified names for all the classes of the specified package by searching for unqualified names as classes qualified by the given package. The package can be a java package, an inetsoft package, or any other package beginning with `com` or `org`.

Example

```
importPackage(java.awt);
```

```
font = new Font("Verdana", Font.ITALIC, 24);
foreground = Color.red;
```

JS.2 String Object Functions

String is a built-in type in JavaScript. It holds a sequence of characters in the Unicode character set. A string literal is created using a double quote or a single quote strings.

```
var txt = 'Hello world'; // or "Hello world"
```

Two strings can be concatenated by using the '+' operator, as well as by using the `concat` function. For example:

```
var txt = "Hello " + 'world';

field['Salesperson.First Name'].concat("
",field['Salesperson.Last Name'])
```

LENGTH

The length property of a string returns the number of characters in the string.

CONCAT(STRING2[,STRING3][,STRING4][,...])

Concatenate a given string with one or more additional strings.

Examples

```
var txt = 'Hello'.concat(' ','and good morning.')

field['Salesperson.First
Name'].concat(" ",field['Salesperson.Last Name'])
```

SUBSTRING(INDEXA, INDEXB)

`indexA` is the starting position of the substring in the original string and `indexB` is the position following the last character in the substring. The index is zero based.

CHARAT(INDEX)

Return the character at the specified position in the string.

CHARCODEAT(INDEX)

Return the character code of the character at the specified position in the string.

INDEXOF(SEARCHVALUE[, FROMINDEX])

Search for the search value in the string and return the position of the sub-string, or –1 if the search value is not found. The optional `fromIndex` specifies the starting position of the search.

LASTINDEXOF(SEARCHVALUE[, FROMINDEX])

Search for the last occurrence of the search value and return the position or –1 if not found. The optional `fromIndex` specifies the starting position of the search.

REPLACE(REGEXP, NEWSTR)

Replace a regular expression in a string with the new string.

SEARCH(REGEXP)

Match a regular expression with a string. If successful, it returns the position of the regular expression in the string, or –1 if no match is found.

TOLOWERCASE()

Convert all characters to lowercase.

TOUPPERCASE()

Convert all characters to uppercase.

There are a number of other functions in the string object. Please consult the JavaScript specification for details.

Example

```
"a new string".toUpperCase();
```

SPLIT(SEPARATOR, LIMIT)

Split up the given string into an array of string values using the specified delimiter.

Parameter

Separator	The characters to be used as delimiters when splitting the string. By default, space is used as the delimiter.
Limit	The number of substrings to be returned.

Example

```
"abc,def".split(",",2);
```

SLICE(START, END)

Extract a certain portion of the string.

Parameter

Start	Index of the first character of the desired substring.
End	Index of the character after the last character of the desired substring.

Example

```
"a new string".slice(2, 5)  // returns 'new'
```

JS.3 Number Object Functions

All numeric values are number objects. The values are normally used for computation using language operators such as multiplication, addition, subtraction and division. There are a few useful methods for converting a number to a string.

TOSTRING([RADIX])

Convert a number to a string. The optional radix is a number ranging from 2 to 36.

Example

```
(5).toString(2);   // 101 binary representation
```

TOFIXED(FRACTIONDIGITS)

Convert a number to a fixed decimal number. The `fractionDigits` specifies the number of digits following the decimal point.

Example

```
(1/3).toFixed(2);
```

TOEXPONENTIAL(FRACTIONDIGITS)

Converts a number to be in scientific format. The `fractionDigits` specifies the number of digits following the decimal point.

Example

```
(3251.234).toExponential(2); // returns 3.25e+3
```

TOLOCALESTRING()

Returns the localized value in the string format for the given number

TOPRECISION(PRECISION)

Returns the floor of the given number. The precision specifies the total number of digits including the ones before and after the decimal points.

Example

```
(3.234).toPrecision(3); // returns 3.23
```

JS.4 Date Object Functions

The JavaScript Date object is very similar to Java Date object. It stores the time and date and can be used to manipulate dates and perform date conversions. There are several ways to create a date object. Invoked without arguments, the new date object contains the time and date of its creation:

```
var now = new Date()  // Thu Jun 12 15:14:51 EDT 2008
```

You can also create a Date object by specifying elapsed milliseconds from January 1, 1970:

```
var due_date = new Date(800000000000);
// Tue May 09 02:13:20 EDT 1995
```

Alternatively, create the Date object from individual date components:

```
var due_date = new Date(1988,0,10);
// Sun Jan 10 00:00:00 EST 1988
```

There are a number of methods in the Date object for manipulating date fields. Since the scripts in Style Intelligence are primarily used to work with Java objects, it is more common for a script to work with the java.util.Date object. The use of Java objects is covered in Appendix JS.9, *Java Objects (LiveConnect)*.

GETMILLISECONDS()

Return the milliseconds from epoch.

TOSTRING()

Return a string representation of the date.

TODATESTRING()

Return a string representation of the date portion of the date.

TOTIMESTRING()

Return a string representation of the time portion of the date.

Examples

```
(new Date()).toTimeString();
```

DATEADD(INTERVAL, AMOUNT, DATE)

Returns a new date that is created by adding the specified amount of time to the specified interval of the given date.

Parameters

interval	The date interval to be modified.
amount	The amount of time to be added to the interval
date	The date to be modified

Example

```
dateAdd('m', 2, new Date())
// newDate is set to be 2 months from the current date.
```

All the valid date interval strings are listed in the table below.

INTERVAL STRINGS	DATE INTERVAL IT REPRESENTS
yyyy	Year
q	Quarter
m	Month
y	Day of the year
d	Day of the month
w	Day of the week
ww	Week of the year
h	Hour
n	Minute
s	Second

DATEDIFF(INTERVAL, DATE1, DATE2)

Return the difference in the interval between the two given dates. For the valid interval values, please refer to the table under dateAdd() function discussion.

Parameters

interval	The date interval for which the difference is to be calculated.
date1, date2	The dates to be compared.

Example

```
dateDiff('m', new Date(), dateAdd('m', 2, new Date()));
// returns 2.
```

DATEPART(INTERVAL, DATE)

Extracts the specified date interval. For the valid interval values, please refer to the table under the dateAdd() function discussion.

Parameters

interval	The interval value to be extracted.
date	The date from which the interval is to be

extracted.

Example

```
datePart('m', new Date()); // returns current month
```

JS.5 Array Object Functions

JavaScript arrays are a non-fixed length arrays. A constant array can be created using an array literal:

```
states = ["NJ", "NY", "CA"];
```

This is equivalent to the array constructor:

```
states = new Array("NJ", "NY", "CA");
```

The length of the array is dynamically adjusted when new items are added to an array. If the newly added item leaves a gap in the existing array, the items in the gap have an 'undefined' value.

```
states[50] = 'FL';
```

The length property of an array object returns the current array length.

```
for(var i = 0; i < states.length; i++) {
  if(states[i] == 'NJ') {
  ...
  }
}
```

Array objects also have functions to merge, slice, reverse, etc.

CONCAT(ARRAY1, ARRAY2, …, ARRAYN)

Concatenate the arrays into one array.

```
num1 = [1,2,3]
num2 = [4,5,6]
num3 = [7,8,9]
nums = num1.concat(num2,num3)
```

SLICE(BEGIN[, END])

Return a slice of the array as a new array.

SORT([COMPAREFUNCTION])

Sort the array items according to a compare function. If the function is not supplied, all items are sorted as strings. A compare function is a function that accepts two parameters. It returns a positive number if the first parameter is greater than the second parameter, zero if the two parameters are the same and a negative number if the first parameter is less than the second parameter.

Examples

```
function compareInt(a, b) {
  return a - b;
}
array1.sort(compareInt);
```

JOIN(SEPARATOR)

Returns a string containing all the elements of the array separated by the specified separator.

Example

```
['A', 'B', 'C'].join('&'); // returns "A&B&C"
```

REVERSE()

Reverses the order of the elements in the array.

Example

```
['A', 'B', 'C'].reverse(); // returns ['C', 'B', 'A']
```

JS.6 Math Object Functions

The Math object is a top-level object, and has some numerical constants as properties (e.g. Math.PI returns 3.141592653589793). The complete list of constants is: E, LN10, LN2, LOG10E, LOG2E, PI, SQRT1_2, SQRT2.

Math also provides many static functions. They can be accessed without an instance of the object.

ABS(NUMBER)

Return the absolute value.

Example

```
Math.abs(-234.45);  // returns 234.45
```

ACOS(NUMBER)

Return the inverse cosine (in radians) of a number between 0 and 1.

Example

```
Math.acos(0);  // returns 1.570796 (90 degrees)
```

ASIN(NUMBER)

Return the inverse sine (in radians) of a number between 0 and 1.

Example

```
Math.asin(1);  // returns 1.570796 (90 degrees)
```

ATAN(NUMBER)

Return the inverse tangent (in radians) of a number.

Example

```
Math.atan(1);  // returns 0.785398 (45 degrees)
```

CEIL(NUMBER)

Return the smallest integer greater than or equal to a number.

Example

```
Math.ceil(15.2);  // returns 16
```

COS(NUMBER)

Return the cosine (in radians) of a number.

Example

```
Math.cos(3.14159);  // returns -0.9999
```

EXP(NUMBER)

Return the value of e raised to the power of number.

Example

```
Math.exp(3);  // returns 20.085
```

FLOOR(NUMBER)

Return the largest integer less than or equal to a number.

Example

```
Math.floor(15.2);  // returns 15
```

LOG(NUMBER)

Return a natural logarithm (base *e*) of a number.

Example

```
Math.log(2.71828);  // returns 1
```

MAX(X, Y)

Return the maximum number, X or Y.

Example

```
Math.max(2,3);  // returns 3
```

MIN(X, Y)

Return the minimum number, X or Y.

Example

```
Math.min(2,3);  // returns 2
```

POW(BASE, EXPONENT)

Raise the base to the exponent power.

Example

```
Math.pow(2,3);  // returns 8 (2^3)
```

RANDOM()

Return a random number between 0 and 1.

Example

```
Math.random();  // returns 0.917008, for example
```

ROUND(NUMBER)

Return the value of a number rounded to the nearest integer.

Example

```
Math.round(15.8);  // returns 16
```

SIN(NUMBER)

Return the sine of an angle in radians

Example

```
Math.sin(pi()/4);  // returns 0.7071
```

SQRT(NUMBER)

Return the square root of a number.

Example

```
Math.sqrt(36.89);  // returns 6.0737
```

TAN(NUMBER)

Return the tangent of an angle in radians

Example

```
Math.tan(0.785398);  // returns 1 tan(45^o)
```

JS.7 CALC Object Functions

The CALC object contains implementations of Excel-like spreadsheet functions. All these functions are static and can be used without an instance of the CALC object.

```
var avg = CALC.average([1,2,3]);
```

Many functions include an optional parameter `basis` to indicate the day count convention:

- 0: US 30/360 (default)
- 1: Actual/Actual
- 2: Actual/360
- 3: Actual/365
- 4: European 30/360

JS.7.1 Date and Time Functions

The CALC object provides the date and time functions listed below.

CALC.DATEVALUE(DATE)

Returns the number of days between the epoch 12/31/1899 and parameter `date`.

Parameters

date	a Date object

Example

```
CALC.datevalue(new Date(1900,0,2));
// returns 2
```

CALC.DAY(DATE)

Returns the day of the month corresponding to the Date object in parameter `date`.

Parameters

date	a Date object

Example

```
CALC.day(new Date());
// returns 28 (Assuming date was Dec 28, 2005)
```

CALC.DAYS360(START_DATE, END_DATE[, METHOD])

Return the number of days between the `start_date` and the `end_date`, using 12 months of 30 days. Set `method` to 'false' to use the U.S. NASD method (default); set `method` to 'true' to use the European method.

Parameters

start_date	Date object: date from which to begin counting
end_date	Date object: date at which to end counting
method	Boolean: false for US, true for European

Example

```
CALC.days360(new Date(2005,1,21), new Date(2006,0,18),
false);
// returns 327
```

CALC.EDATE(DATE, MONTHS)

Returns a date that is the specified number of `months` before or after the `date`. A negative value for `months` represents a date prior to the specified `date`.

Parameters

date Date object: date from which to begin counting
months Positive integer: Number of months to count forward
Negative integer: Number of months to count backward

Example

```
// Add one month to Jan 14, 2000:
CALC.edate(new Date(2000,0,14), 1); // returns Feb 14, 2000
```

CALC.EOMONTH(DATE, MONTHS)

Returns a date corresponding to the last day of the month that is a specified number of `months` before or after the given `date`. A negative value for `months` represents a date prior to the specified `date`.

Parameters

date Date object: date from which to begin counting
months Positive integer: Number of months to count forward
Negative integer: Number of months to count backward

Example

```
// Last day of month for month following Jan 14, 2000:
CALC.eomonth(new Date(2000,0,14), 1);
// returns 2/29/2000
```

CALC.FISCALYEAR(DATE, STARTMONTH [,STARTDAY,TIMEZONE])

For a fiscal calendar defined by an arbitrary start date, returns the fiscal year corresponding to the specified `date` as a number, e.g., “2014”. The `startMonth` and optional `startDay` determine the starting point of the fiscal year.

Parameters

date Date object to be converted to fiscal year.
startMonth Integer specifying the starting month of the fiscal calendar.
startDay Integer specifying the starting day of the fiscal calendar (defaults to 1).

timeZone Time zone for fiscal year start date, a valid time zone identifier ("GMT", "GMT-8", "America/New_York"). Default is the JVM system default time zone.

Example

```
// Today's year in fiscal year beginning April 16:
CALC.fiscalyear(CALC.today(), 4, 16)
```

CALC.FISCALYEAR445(DATE, STARTYEAR, STARTMONTH, STARTDAY, LEAPYEARS[,TIMEZONE])

For a 4-4-5 fiscal calendar, returns the fiscal year corresponding to the specified `date` as a number, e.g., "2014". The `startYear`, `startMonth`, and `startDay` determine the starting point of the fiscal calendar. Leap years must be explicitly specified by the array `leapYears`.

Parameters

date Date object to be converted to fiscal year.
startYear Integer specifying the starting year of the fiscal calendar.
startMonth Integer specifying the starting month of the fiscal calendar.
startDay Integer specifying the starting day of the fiscal calendar.
leapYears Array of years that contain 53 weeks.
timeZone Time zone for fiscal year start date, a valid time zone identifier ("GMT", "GMT-8", "America/New_York"). Default is the JVM system default time zone.

Example

```
// Today's year in fiscal calendar beginning Jan 30, 2000
// with leap years in 2000, 2006, 2012, and 2017:
CALC.fiscalyear445(CALC.today(), 2000, 1, 30, [2000, 2006, 2012, 2017])
```

In a 4-4-5 fiscal calendar, a typical year has 52 weeks totalling 364 days:

- The first month of each quarter is 4 weeks
- The second month of each quarter is 4 weeks
- The third month of each quarter is 5 weeks

Every few years (as specified by array `leapYears`), the fiscal calendar requires a 53-week year in order to reconcile with the solar year. In any such leap year, the final month of the last quarter has 6 weeks.

CALC.FISCALYEAR454(DATE, STARTYEAR, STARTMONTH, STARTDAY, LEAPYEARS[,TIMEZONE])

For a 4-5-4 fiscal calendar, returns the fiscal year corresponding to the specified `date` as a number, e.g., "2014". The `startYear`, `startMonth`, and `startDay` determine the starting point of the fiscal calendar. Leap years must be explicitly specified by the array `leapYears`.

Parameters

```
date          Date object to be converted to fiscal year.
startYear     Integer specifying the starting year of the
fiscal calendar.
startMonth    Integer specifying the starting month of the
fiscal calendar.
startDay      Integer specifying the starting day of the
fiscal calendar.
leapYears     Array of years that contain 53 weeks.
timeZone      Time zone for fiscal year start date, a valid
time zone identifier ("GMT", "GMT-8", "America/New_York").
Default is the JVM system default time zone.
```

Example

```
// Today's year in fiscal calendar beginning Jan 30, 2000
// with leap years in 2000, 2006, 2012, and 2017:
CALC.fiscalyear454(CALC.today(), 2000, 1, 30, [2000, 2006,
2012, 2017])
```

In a 4-5-4 fiscal calendar, a typical year has 52 weeks totalling 364 days:

- The first month of each quarter is 4 weeks
- The second month of each quarter is 5 weeks
- The third month of each quarter is 4 weeks

Every few years (as specified by array `leapYears`), the fiscal calendar requires a 53-week year in order to reconcile with the solar year. In any such leap year, the final month of the last quarter has 5 weeks.

CALC.FISCALYEAR544(DATE, STARTYEAR, STARTMONTH, STARTDAY, LEAPYEARS[,TIMEZONE])

For a 5-4-4 fiscal calendar, returns the fiscal year corresponding to the specified `date` as a number, e.g., "2014". The `startYear`, `startMonth`, and `startDay` determine the starting point of the fiscal calendar. Leap years must be explicitly specified by the array `leapYears`.

Parameters

```
date          Date object to be converted to fiscal year.
startYear     Integer specifying the starting year of the
fiscal calendar.
startMonth    Integer specifying the starting month of the
fiscal calendar.
startDay      Integer specifying the starting day of the
fiscal calendar.
leapYears     Array of years that contain 53 weeks.
timeZone      Time zone for fiscal year start date, a valid
time zone identifier ("GMT", "GMT-8", "America/New_York").
Default is the JVM system default time zone.
```

Example

```
// Today's year in fiscal calendar beginning Jan 30, 2000
// with leap years in 2000, 2006, 2012, and 2017:
CALC.fiscalyear544(CALC.today(), 2000, 1, 30, [2000, 2006,
2012, 2017])
```

In a 5-4-4 fiscal calendar, a typical year has 52 weeks totalling 364 days:

- The first month of each quarter is 5 weeks
- The second month of each quarter is 4 weeks
- The third month of each quarter is 4 weeks

Every few years (as specified by array `leapYears`), the fiscal calendar requires a 53-week year in order to reconcile with the solar year. In any such leap year, the final month of the last quarter has 5 weeks.

CALC.FISCALQUARTER(DATE, STARTMONTH [,STARTDAY,TIMEZONE])

For a fiscal calendar defined by an arbitrary start date, returns the fiscal quarter corresponding to the specified `date` as a number, e.g., “4”. The `startMonth` and optional `startDay` determine the starting point of the fiscal year.

Parameters

```
date          Date object to be converted to fiscal quarter.
startMonth    Integer specifying the starting month of the
fiscal calendar.
startDay      Integer specifying the starting day of the
fiscal calendar (defaults to 1).
timeZone      Time zone for fiscal year start date, a valid
time zone identifier ("GMT", "GMT-8", "America/New_York").
Default is the JVM system default time zone.
```

Example

```
// Today’s quarter in fiscal year beginning April 16:
CALC.fiscalquarter(CALC.today(), 4, 16)
```

If the start of the fiscal year is on the 29th, 30th, or 31st day of a month, the actual last day of the month is used as the month boundary. For example, if you specify March 31 as the start of the fiscal year, this designates March 31 as the first day of month 1. However, the first day of month 2 *cannot* be April 31 because April has only 30 days. Therefore, the start of month 2 is the actual last day of the month, April 30.

CALC.FISCALQUARTER445(DATE, STARTYEAR, STARTMONTH, STARTDAY, LEAPYEARS[,TIMEZONE])

For a 4-4-5 fiscal calendar, returns the fiscal quarter corresponding to the specified `date` as a number, e.g., “3”. The `startYear`, `startMonth`, and `startDay` determine the starting point of the fiscal calendar. Leap years must be explicitly specified by the array `leapYears`.

Parameters

```
date          Date object to be converted to fiscal quarter.
startYear     Integer specifying the starting year of the
fiscal calendar.
```

startMonth Integer specifying the starting month of the fiscal calendar.
startDay Integer specifying the starting day of the fiscal calendar.
leapYears Array of years that contain 53 weeks.
timeZone Time zone for fiscal year start date, a valid time zone identifier ("GMT", "GMT-8", "America/New_York"). Default is the JVM system default time zone.

Example

```
// Today's quarter in fiscal calendar beginning Jan 30, 2000
// with leap years in 2000, 2006, 2012, and 2017:
CALC.fiscalquarter445(CALC.today(), 2000, 1, 30, [2000,
2006, 2012, 2017])
```

In a 4-4-5 fiscal calendar, a typical year has 52 weeks totalling 364 days:

- The first month of each quarter is 4 weeks
- The second month of each quarter is 4 weeks
- The third month of each quarter is 5 weeks

Every few years (as specified by array `leapYears`), the fiscal calendar requires a 53-week year in order to reconcile with the solar year. In any such leap year, the final month of the last quarter has 6 weeks.

CALC.FISCALQUARTER454(DATE, STARTYEAR, STARTMONTH, STARTDAY, LEAPYEARS[,TIMEZONE])

For a 4-5-4 fiscal calendar, returns the fiscal quarter corresponding to the specified `date` as a number, e.g., "3". The `startYear`, `startMonth`, and `startDay` determine the starting point of the fiscal calendar. Leap years must be explicitly specified by the array `leapYears`.

Parameters

date Date object to be converted to fiscal quarter.
startYear Integer specifying the starting year of the fiscal calendar.
startMonth Integer specifying the starting month of the fiscal calendar.
startDay Integer specifying the starting day of the fiscal calendar.
leapYears Array of years that contain 53 weeks.
timeZone Time zone for fiscal year start date, a valid time zone identifier ("GMT", "GMT-8", "America/New_York"). Default is the JVM system default time zone.

Example

```
// Today's quarter in fiscal calendar beginning Jan 30, 2000
// with leap years in 2000, 2006, 2012, and 2017:
CALC.fiscalquarter454(CALC.today(), 2000, 1, 30, [2000,
2006, 2012, 2017])
```

In a 4-5-4 fiscal calendar, a typical year has 52 weeks totalling 364 days:

- The first month of each quarter is 4 weeks
- The second month of each quarter is 5 weeks
- The third month of each quarter is 4 weeks

Every few years (as specified by array `leapYears`), the fiscal calendar requires a 53-week year in order to reconcile with the solar year. In any such leap year, the final month of the last quarter has 5 weeks.

CALC.FISCALQUARTER544(DATE, STARTYEAR, STARTMONTH, STARTDAY, LEAPYEARS[,TIMEZONE])

For a 5-4-4 fiscal calendar, returns the fiscal quarter corresponding to the specified `date` as a number, e.g., “3”. The `startYear`, `startMonth`, and `startDay` determine the starting point of the fiscal calendar. Leap years must be explicitly specified by the array `leapYears`.

Parameters

```
date          Date object to be converted to fiscal quarter.
startYear     Integer specifying the starting year of the
fiscal calendar.
startMonth    Integer specifying the starting month of the
fiscal calendar.
startDay      Integer specifying the starting day of the
fiscal calendar.
leapYears     Array of years that contain 53 weeks.
timeZone       Time zone for fiscal year start date, a valid
time zone identifier ("GMT", "GMT-8", "America/New_York").
Default is the JVM system default time zone.
```

Example

```
// Today’s quarter in fiscal calendar beginning Jan 30, 2000
// with leap years in 2000, 2006, 2012, and 2017:
CALC.fiscalquarter544(CALC.today(), 2000, 1, 30, [2000,
2006, 2012, 2017])
```

In a 5-4-4 fiscal calendar, a typical year has 52 weeks totalling 364 days:

- The first month of each quarter is 5 weeks
- The second month of each quarter is 4 weeks
- The third month of each quarter is 4 weeks

Every few years (as specified by array `leapYears`), the fiscal calendar requires a 53-week year in order to reconcile with the solar year. In any such leap year, the final month of the last quarter has 5 weeks.

CALC.FISCALMONTH(DATE, STARTMONTH [,STARTDAY,TIMEZONE])

For a fiscal calendar defined by an arbitrary start date, returns the fiscal month corresponding to the specified `date` as a number, e.g., “12”. The

`startMonth` and optional `startDay` determine the starting point of the fiscal year.

Parameters

```
date          Date object to be converted to fiscal month.
startMonth    Integer specifying the starting month of the
fiscal calendar.
startDay      Integer specifying the starting day of the
fiscal calendar (defaults to 1).
timeZone       Time zone for fiscal year start date, a valid
time zone identifier ("GMT", "GMT-8", "America/New_York").
Default is the JVM system default time zone.
```

Example

```
// Today's month in fiscal year beginning April 16:
CALC.fiscalmonth(CALC.today(), 4, 16)
```

If the start of the fiscal year is on the 29th, 30th, or 31st day of a month, the actual last day of the month is used as the month boundary. For example, if you specify March 31 as the start of the fiscal year, this designates March 31 as the first day of month 1. However, the first day of month 2 *cannot* be April 31 because April has only 30 days. Therefore, the start of month 2 is the actual last day of the month, April 30.

CALC.FISCALMONTH445(DATE, STARTYEAR, STARTMONTH, STARTDAY, LEAPYEARS[,TIMEZONE])

For a 4-4-5 fiscal calendar, returns the fiscal month corresponding to the specified `date` as a number, e.g., "8". The `startYear`, `startMonth`, and `startDay` determine the starting point of the fiscal calendar. Leap years must be explicitly specified by the array `leapYears`.

Parameters

```
date          Date object to be converted to fiscal month.
startYear     Integer specifying the starting year of the
fiscal calendar.
startMonth    Integer specifying the starting month of the
fiscal calendar.
startDay      Integer specifying the starting day of the
fiscal calendar.
leapYears     Array of years that contain 53 weeks.
timeZone       Time zone for fiscal year start date, a valid
time zone identifier ("GMT", "GMT-8", "America/New_York").
Default is the JVM system default time zone.
```

Example

```
// Today's month in fiscal calendar beginning Jan 30, 2000
// with leap years in 2000, 2006, 2012, and 2017:
CALC.fiscalmonth445(CALC.today(), 2000, 1, 30, [2000, 2006,
2012, 2017])
```

In a 4-4-5 fiscal calendar, a typical year has 52 weeks totalling 364 days:

- The first month of each quarter is 4 weeks

- The second month of each quarter is 4 weeks
- The third month of each quarter is 5 weeks

Every few years (as specified by array `leapYears`), the fiscal calendar requires a 53-week year in order to reconcile with the solar year. In any such leap year, the final month of the last quarter has 6 weeks.

CALC.FISCALMONTH454(DATE, STARTYEAR, STARTMONTH, STARTDAY, LEAPYEARS[,TIMEZONE])

For a 4-5-4 fiscal calendar, returns the fiscal month corresponding to the specified `date` as a number, e.g., “11”. The `startYear`, `startMonth`, and `startDay` determine the starting point of the fiscal calendar. Leap years must be explicitly specified by the array `leapYears`.

Parameters

```
date          Date object to be converted to fiscal month.
startYear     Integer specifying the starting year of the
fiscal calendar.
startMonth    Integer specifying the starting month of the
fiscal calendar.
startDay      Integer specifying the starting day of the
fiscal calendar.
leapYears     Array of years that contain 53 weeks.
timeZone       Time zone for fiscal year start date, a valid
time zone identifier ("GMT", "GMT-8", "America/New_York").
Default is the JVM system default time zone.
```

Example

```
// Today’s month in fiscal calendar beginning Jan 30, 2000
// with leap years in 2000, 2006, 2012, and 2017:
CALC.fiscalmonth454(CALC.today(), 2000, 1, 30, [2000, 2006,
2012, 2017])
```

In a 4-5-4 fiscal calendar, a typical year has 52 weeks totalling 364 days:

- The first month of each quarter is 4 weeks
- The second month of each quarter is 5 weeks
- The third month of each quarter is 4 weeks

Every few years (as specified by array `leapYears`), the fiscal calendar requires a 53-week year in order to reconcile with the solar year. In any such leap year, the final month of the last quarter has 5 weeks.

CALC.FISCALMONTH544(DATE, STARTYEAR, STARTMONTH, STARTDAY, LEAPYEARS[,TIMEZONE])

For a 5-4-4 fiscal calendar, returns the fiscal month corresponding to the specified `date` as a number, e.g., “11”. The `startYear`, `startMonth`, and `startDay` determine the starting point of the fiscal calendar. Leap years must be explicitly specified by the array `leapYears`.

Parameters

```
date          Date object to be converted to fiscal month.
startYear     Integer specifying the starting year of the
fiscal calendar.
startMonth    Integer specifying the starting month of the
fiscal calendar.
startDay      Integer specifying the starting day of the
fiscal calendar.
leapYears     Array of years that contain 53 weeks.
timeZone       Time zone for fiscal year start date, a valid
time zone identifier ("GMT", "GMT-8", "America/New_York").
Default is the JVM system default time zone.
```

Example

```
// Today's month in fiscal calendar beginning Jan 30, 2000
// with leap years in 2000, 2006, 2012, and 2017:
CALC.fiscalmonth544(CALC.today(), 2000, 1, 30, [2000, 2006,
2012, 2017])
```

In a 5-4-4 fiscal calendar, a typical year has 52 weeks totalling 364 days:

- The first month of each quarter is 5 weeks
- The second month of each quarter is 4 weeks
- The third month of each quarter is 4 weeks

Every few years (as specified by array `leapYears`), the fiscal calendar requires a 53-week year in order to reconcile with the solar year. In any such leap year, the final month of the last quarter has 5 weeks.

CALC.FISCALWEEK(DATE, STARTMONTH [,STARTDAY,TIMEZONE])

For a fiscal calendar defined by an arbitrary start date, returns the fiscal week corresponding to the specified `date` as a number, e.g., "46". The `startMonth` and optional `startDay` determine the starting point of the fiscal year.

Parameters

```
date          Date object to be converted to fiscal week.
startMonth    Integer specifying the starting month of the
fiscal calendar.
startDay      Integer specifying the starting day of the
fiscal calendar (defaults to 1).
timeZone       Time zone for fiscal year start date, a valid
time zone identifier ("GMT", "GMT-8", "America/New_York").
Default is the JVM system default time zone.
```

Example

```
// Today's week in fiscal year beginning April 16:
CALC.fiscalweek(CALC.today(), 4, 16)
```

CALC.FISCALWEEK445(DATE, STARTYEAR, STARTMONTH, STARTDAY, LEAPYEARS[,TIMEZONE])

For a 4-4-5 fiscal calendar, returns the fiscal week corresponding to the specified `date` as a number, e.g., “42”. The `startYear`, `startMonth`, and `startDay` determine the starting point of the fiscal calendar. Leap years must be explicitly specified by the array `leapYears`.

Parameters

```
date          Date object to be converted to fiscal week.
startYear     Integer specifying the starting year of the
fiscal calendar.
startMonth    Integer specifying the starting month of the
fiscal calendar.
startDay      Integer specifying the starting day of the
fiscal calendar.
leapYears     Array of years that contain 53 weeks.
timeZone       Time zone for fiscal year start date, a valid
time zone identifier ("GMT", "GMT-8", "America/New_York").
Default is the JVM system default time zone.
```

Example

```
// Today’s week in fiscal calendar beginning Jan 30, 2000
// with leap years in 2000, 2006, 2012, and 2017:
CALC.fiscalweek445(CALC.today(), 2000, 1, 30, [2000, 2006,
2012, 2017])
```

In a 4-4-5 fiscal calendar, a typical year has 52 weeks totalling 364 days:

- The first month of each quarter is 4 weeks
- The second month of each quarter is 4 weeks
- The third month of each quarter is 5 weeks

Every few years (as specified by array `leapYears`), the fiscal calendar requires a 53-week year in order to reconcile with the solar year. In any such leap year, the final month of the last quarter has 6 weeks.

CALC.FISCALWEEK454(DATE, STARTYEAR, STARTMONTH, STARTDAY, LEAPYEARS[,TIMEZONE])

For a 4-5-4 fiscal calendar, returns the fiscal week corresponding to the specified `date` as a number, e.g., “42”. The `startYear`, `startMonth`, and `startDay` determine the starting point of the fiscal calendar. Leap years must be explicitly specified by the array `leapYears`.

Parameters

```
date          Date object to be converted to fiscal week.
startYear     Integer specifying the starting year of the
fiscal calendar.
startMonth    Integer specifying the starting month of the
fiscal calendar.
startDay      Integer specifying the starting day of the
fiscal calendar.
```

leapYears Array of years that contain 53 weeks.
timeZone Time zone for fiscal year start date, a valid time zone identifier ("GMT", "GMT-8", "America/New_York"). Default is the JVM system default time zone.

Example

```
// Today's week in fiscal calendar beginning Jan 30, 2000
// with leap years in 2000, 2006, 2012, and 2017:
CALC.fiscalweek454(CALC.today(), 2000, 1, 30, [2000, 2006,
2012, 2017])
```

In a 4-5-4 fiscal calendar, a typical year has 52 weeks totalling 364 days:

- The first month of each quarter is 4 weeks
- The second month of each quarter is 5 weeks
- The third month of each quarter is 4 weeks

Every few years (as specified by array `leapYears`), the fiscal calendar requires a 53-week year in order to reconcile with the solar year. In any such leap year, the final month of the last quarter has 5 weeks.

CALC.FISCALWEEK544(DATE, STARTYEAR, STARTMONTH, STARTDAY, LEAPYEARS[,TIMEZONE])

For a 5-4-4 fiscal calendar, returns the fiscal week corresponding to the specified `date` as a number, e.g., "42". The `startYear`, `startMonth`, and `startDay` determine the starting point of the fiscal calendar. Leap years must be explicitly specified by the array `leapYears`.

Parameters

date Date object to be converted to fiscal week.
startYear Integer specifying the starting year of the fiscal calendar.
startMonth Integer specifying the starting month of the fiscal calendar.
startDay Integer specifying the starting day of the fiscal calendar.
leapYears Array of years that contain 53 weeks.
timeZone Time zone for fiscal year start date, a valid time zone identifier ("GMT", "GMT-8", "America/New_York"). Default is the JVM system default time zone.

Example

```
// Today's week in fiscal calendar beginning Jan 30, 2000
// with leap years in 2000, 2006, 2012, and 2017:
CALC.fiscalweek544(CALC.today(), 2000, 1, 30, [2000, 2006,
2012, 2017])
```

In a 5-4-4 fiscal calendar, a typical year has 52 weeks totalling 364 days:

- The first month of each quarter is 5 weeks
- The second month of each quarter is 4 weeks
- The third month of each quarter is 4 weeks

Every few years (as specified by array `leapYears`), the fiscal calendar requires a 53-week year in order to reconcile with the solar year. In any such leap year, the final month of the last quarter has 5 weeks.

CALC.HOUR(DATE)

Returns the hour of the day for the given `date` object.

Parameters

`date` Date object

Example

```
CALC.hour(new Date(2000,0,14,7,15,23));
// returns 7
```

CALC.MINUTE(DATE)

Returns the minute of the hour for the given `date` object.

Parameters

`date` Date object

Example

```
CALC.minute(new Date(2000,0,14,7,15,23));
// returns 15
```

CALC.MONTH(DATE)

Returns the numeric month for the given `date` object. (January has value 1.)

Parameters

`date` Date object

Example

```
CALC.month(new Date(2000,1,21));
// returns 2 (Feb)
```

CALC.MONTHNAME(DATE)

Returns the name of the month for the given `date` object (e.g. “March”).

Parameters

`date` Date object

Example

```
CALC.monthname(new Date(1978,1,21));
// returns February
```

CALC.NETWORKDAYS(START_DATE, END_DATE[, HOLIDAY_ARRAY])

Returns the total number of working days between `start_date` and `end_date`. If either `start_date` or `end_date` is itself a working day, that day

is also included in the total. `Holiday_array` is an optional array of dates to be excluded from the working day count.

Parameters

```
start_date    Date object: date on which to begin counting
end_date      Date object: date on which to end counting
holiday_array Array of Dates to exclude from working day
count
```

Example

```
// Find working days from Feb. 21, 1978 until today:
CALC.networkdays(new Date(1978,1,21), CALC.today());
// returns 9463 (assuming today is May 29, 2014)
```

CALC.NOW()

Returns the current date and time.

Example

```
CALC.now();
// returns current date and time (e.g., Thu May 29 15:45:59
EDT 2014)
```

CALC.QUARTER(DATE)

Returns the quarter (1, 2, 3, or 4) corresponding to the given `date`.

Parameters

```
date          Date object
```

Example

```
CALC.quarter(new Date(2005, 11, 28));
// returns 4
```

CALC.SECOND(DATE)

Returns the seconds corresponding to the given `date`.

Parameters

```
date          Date object
```

Example

```
CALC.second(CALC.today());
// returns 37 if CALC.today() is Thu May 29 16:12:37 EDT
2014
```

CALC.TIME(HOUR, MINUTE, SECOND)

Returns a decimal between 0 and 1 representing the specified time as a fraction of the 24-hour day. For example, time 0:00 has value 0.0, while time 12:00 has value 0.5.

Parameters

hour	Integer value for hour (24-hour time)
minute	Integer value for minute
second	Integer value for second

Example

```
CALC.time(0,0,0);     // returns 0.0 (12am)
CALC.time(12,0,0);    // returns 0.5 (12pm)
CALC.time(23,59,59)); // returns 0.9999884366989136
CALC.time(5,30,40);   // returns 0.22962963581085205
```

CALC.TIMEVALUE(DATE)

Returns a decimal between 0 and 1 representing the time in the specified `date` object as a fraction of the 24-hour day. For example, time 0:00 has value 0.0, while time 12:00 has value 0.5.

Parameters

date	Date object

Example

```
CALC.timevalue(new Date(2014,0,1,0,0,0)); // returns 0.0
CALC.timevalue(new Date(2014,0,1,12,0,0)) // returns 0.5
CALC.timevalue(new Date(2005,1,2,6,0,0)); // returns 0.25
```

CALC.TODAY()

Returns a date object containing the current date and time.

Example

```
CALC.today();
// returns Tue Mar 17 17:43:02 EDT 2009 (for example)
```

CALC.WEEKDAY(DATE [,NUMBER])

Returns an integer representing the day of the week corresponding to the specified `date`. Optional parameter `number` determines the value at which indexing starts, and can have the following values:

- 1: Returns the day as a number representing Sunday (1) through Saturday (7). This is the default.
- 2: Returns the day as a number representing Monday (1) through Sunday (7).
- 3: Returns the day as a number representing Monday (0) through Sunday (6).

Parameters

date	Date object
number	A code (1,2,3) to specify starting index (see above)

Example

```
CALC.weekday(new Date(2006,0,18),2);
// returns 3 (Wednesday)
```

CALC.WEEKDAYNAME(DATE)

Returns the full name for the day of the week corresponding to the given `date` (e.g., “Monday”).

Parameters

date	Date object

Example

```
CALC.weekdayname(new Date(2006,0,18));
// returns Wednesday
```

CALC.WEEKNUM(DATE)

Returns the week number within the year for the given `date`. (The first week of the year has index 1.)

Parameters

date	Date object

Example

```
CALC.weeknum(new Date(2006,0,18));
// returns 3
```

CALC.WORKDAY(DATE, DAYS, [HOLIDAY_ARRAY])

Returns a date representing the first workday following a period of working days, specified by `days`, that begin on `date`. Optional parameter `Holiday_array` specifies a list of dates to be excluded from the working day count.

Parameters

date	Date object: date on which to begin counting
days	Number of working days to count
holiday_array	Array of Dates to exclude from working day count

Example

```
// First working day following six days from May 20, 2014:
CALC.workday(new Date(2014,4,20), 6) // returns Wed May 28
```

CALC.YEAR(DATE)

Returns the year corresponding to the given `date`.

Parameters

date	Date object

Example

```
CALC.year(new Date(2005,11,1));
// returns 2005
```

CALC.YEARFRAC(START_DATE, END_DATE [, BASIS])

Returns a decimal value between 0 and 1 representing the number of days between `start_date` and `end_date` as a fraction of the year.

Parameters

start_date	Date object: date on which to begin counting
end_date	Date object: date on which to end counting
basis	Day count convention code (see below)

Optional parameter `basis` indicates the day count convention:

- 0: US 30/360 (default)
- 1: Actual/Actual
- 2: Actual/360
- 3: Actual/365
- 4: European 30/360

Example

```
CALC.yearfrac(new Date(2014,0,1), new Date(2015,0,1));
// returns 1.0

CALC.yearfrac(new Date(2005,0,1), new Date(2005,5,30));
// returns 0.4972222149372101
```

JS.7.2 Financial Functions

The CALC object provides the financial functions listed below.

CALC.ACCRINT(ISSUE, INTEREST, SETTLEMENT, RATE, PAR, FREQ [, BASIS])

Returns the accrued interest for a security that pays periodic interest.

Parameters

issue	Date object: The security's issue date
interest	Date object: The security's first interest date
settlement	Date object: The security's settlement date
rate	The security's annual coupon rate
par	The security's par value
freq	The number of coupon payments per year
basis	Day count convention code (see below)

Optional parameter `basis` indicates the day count convention:

- 0: US 30/360 (default)
- 1: Actual/Actual

- 2: Actual/360
- 3: Actual/365
- 4: European 30/360

Example

Interest accrued for a $1000 par value security with a 10% annual rate and bi-annual payment, issue date of March 1, 2008, first interest date of August 31, 2008, and settlement date of May 1, 2008.

```
CALC.accrint(new Date(2008,2,1), new Date(2008,7,31), new
Date(2008,4,1), 0.1, 1000, 2);
// returns 16.666666666666664
```

CALC.ACCRINTM(ISSUE, MATURITY, RATE, PAR, [BASIS])

Returns the accrued interest for a security that pays interest at maturity.

Parameters

issue	Date object: The security's issue date
maturity	Date object: The security's maturity date
settlement	Date object: The security's settlement date
rate	The security's annual coupon rate
par	The security's par value
basis	Day count convention code (see below)

Optional parameter `basis` indicates the day count convention:

- 0: US 30/360 (default)
- 1: Actual/Actual
- 2: Actual/360
- 3: Actual/365
- 4: European 30/360

Example

Interest accrued for a $1000 par value security with a 10% annual rate, issue date of April 1, 2008, maturity date of June 15, 2008, using an Actual/365 counting method.

```
CALC.accrintm(new Date(2008,3,1), new Date(2008,5,15), 0.1,
1000, 3);
// returns 20.54794520547945
```

CALC.AMORDEGRC(COST, DATE_PURCHASED, FIRST_PERIOD, SALVAGE, PERIOD, RATE, [BASIS])

Returns the depreciation of an asset for accounting year `period` on a prorated basis according to the degressive method of the French accounting system. This function is similar to the amorlinc() function except that the depreciation coefficient is calculated based on the life of the asset:

- 0-2 years = 1
- 3-4 years = 1.5
- 5-6 years = 2
- > 6 years = 2.5

Parameters

cost	Initial cost of the asset
date_purchased	Date object: Asset purchase date
first_period	Date object: End of the first period
salvage	Salvage value at end of asset life
period	Period (year) for which to return depreciation
rate	Rate of depreciation
basis	Day count convention code (see below)

Optional parameter `basis` indicates the day count convention:

- 0: US 30/360 (default)
- 1: Actual/Actual
- 2: Actual/360
- 3: Actual/365
- 4: European 30/360

Example

Depreciation over time of a $2400 asset purchased on August 19, 2008, with first depreciation period ending December 31, 2008, end-of-life salvage value of $300, 15% rate of depreciation, and Actual/360 counting method.

```
var deprec = 0;
for(var period =0; period < 7; period++) {
  deprec = CALC.amordegrc(2400, new Date(2008,7,19), new
Date(2008,11,31), 300, period, 0.15, 2);
  log('Period ' + period + ': Depreciation of ' + deprec);
}

// Returns:
// Period 0: Depreciation of 335
// Period 1: Depreciation of 774
// Period 2: Depreciation of 484
// Period 3: Depreciation of 303
// Period 4: Depreciation of 189
// Period 5: Depreciation of 158
// Period 6: Depreciation of 0
```

Depreciation during the first (fractional) period is given by:

```
cost * rate/100 *
yearfrac(date_purchased,first_period,basis)
```

Depreciation during (whole) periods 1 and 2 is given by:

```
rate * cost
```

Depreciation during the final (fractional) period is the difference between the asset's value in the previous period and the asset's salvage value. The function returns 0 for periods after which the asset has already reached its salvage value.

CALC.AMORLINC(COST, DATE_PURCHASED, FIRST_PERIOD, SALVAGE, PERIOD, RATE, [BASIS])

Returns the depreciation of an asset for accounting year `period` on a prorated basis according to the linear depreciation method of the French accounting system. This function is the same as the amordegrc() function except in calculation of the depreciation coefficients.

Parameters

cost Initial cost of the asset
date_purchased Date object: Asset purchase date
first_period Date object: End of the first depreciation period
salvage Salvage value at end of asset life
period Period (year index) for which to return depreciation. First period, from date_purchased to first_period, is index 0.
rate Rate of depreciation
basis Day count convention code (see above)

Optional parameter `basis` indicates the day count convention:

- 0: US 30/360 (default)
- 1: Actual/Actual
- 2: Actual/360
- 3: Actual/365
- 4: European 30/360

Example

Depreciation over time on a $2400 asset purchased on August 19, 2008, with first depreciation period ending December 31, 2008, end-of-life salvage value of $300, 15% rate of depreciation, and Actual/360 counting method.

```
var deprec = 0;
for(var period = 0; period < 10; period++) {
  deprec = CALC.amorlinc(2400, new Date(2008,7,19), new
Date(2008,11,31), 300, period, 0.15, 2);
  log('Period ' + period + ': Depreciation of ' + deprec);
}

// Returns:
// Period 0: Depreciation of 133.99999737739563
// Period 1: Depreciation of 360
// Period 2: Depreciation of 360
```

```
// Period 3: Depreciation of 360
// Period 4: Depreciation of 360
// Period 5: Depreciation of 360
// Period 6: Depreciation of 166.00000262260437
// Period 7: Depreciation of 0
// Period 8: Depreciation of 0
// Period 9: Depreciation of 0
```

Depreciation during the first (fractional) period is given by:

```
cost * rate/100 *
yearfrac(date_purchased,first_period,basis)
```

Depreciation during subsequent (whole) periods is given by:

```
rate * cost
```

Depreciation during the final (fractional) period is the difference between the asset's value in the previous period and the asset's salvage value. The function returns 0 for periods after which the asset has already reached its salvage value.

CALC.COUPDAYBS(SETTLEMENT, MATURITY, FREQUENCY [,BASIS])

Returns the number of days from the most recent coupon date (preceding settlement) to the settlement date.

Parameters

settlement	Date object: Security's settlement date
maturity	Date object: Security's maturity date (date when security expires)
frequency	Number of coupon payments per year (1=annual, 2=semi-annual, 4=quarterly)
basis	Day count convention code (see below)

Optional parameter `basis` indicates the day count convention:

- 0: US 30/360 (default)
- 1: Actual/Actual
- 2: Actual/360
- 3: Actual/365
- 4: European 30/360

Example

Number of days from coupon date to settlement date, where settlement date is January 25, 2007, payments are made twice per year, maturity date is November 15, 2008, and Actual/Actual counting convention is used.

```
CALC.coupdaybs(new Date(2007,0,25), new Date(2008,10,15),
2, 1);
// returns 71
```

With semi-annual payments, the coupon dates in this example are May 15 and November 15, hence the function returns the number of days from coupon date November 15, 2006 to settlement date January 25, 2007, equivalent to

```
dateDiff('d',new Date(2006,10,15), new Date(2007,0,25))
```

CALC.COUPDAYS(SETTLEMENT, MATURITY, FREQUENCY, [BASIS])

Returns the number of days in the coupon period that contains the settlement date.

Parameters

settlement	Date object: Security's settlement date
maturity	Date object: Security's maturity date (date when security expires)
frequency	Number of coupon payments per year (1=annual, 2=semi-annual, 4=quarterly)
basis	Day count convention code (see below)

Optional parameter `basis` indicates the day count convention:

- 0: US 30/360 (default)
- 1: Actual/Actual
- 2: Actual/360
- 3: Actual/365
- 4: European 30/360

Example

Number of days in coupon period that includes settlement date of January 25, 2007, where payments are made twice per year, maturity date is November 15, 2008, and Actual/Actual counting convention is used.

```
CALC.coupdays(new Date(2007,0,25), new Date(2008,10,15), 2,
1);
// returns 181
```

With semi-annual payments, the coupon dates in this example are May 15 and November 15, hence the function returns the number of days from coupon date November 15, 2006 to coupon date May 15, 2007 (the span that includes the settlement date).

CALC.COUPDAYSNC(SETTLEMENT, MATURITY, FREQUENCY, [BASIS])

Returns the number of days from the settlement date to the next coupon date.

Parameters

```
settlement          Date object: Security's settlement
date
maturity            Date object: Security's maturity date
(date when security expires)
frequency           Number of coupon payments per year
(1=annual, 2=semi-annual, 4=quarterly)
basis               Day count convention code (see below)
```

Optional parameter `basis` indicates the day count convention:

- 0: US 30/360 (default)
- 1: Actual/Actual
- 2: Actual/360
- 3: Actual/365
- 4: European 30/360

Example

Number of days from the settlement date of January 25, 2007 to the first coupon date, where payments are made twice per year, maturity date is November 15, 2008, and Actual/Actual counting convention is used.

```
CALC.coupdaysnc(new Date(2007,0,25), new Date(2008,10,15),
2, 1);
// returns 110
```

With semi-annual payments, the coupon dates in this example are May 15 and November 15, hence the function returns the number of days from the settlement date of January 25, 2007 to the first coupon date of May 15, 2007.

CALC.COUPNCD(SETTLEMENT, MATURITY, FREQUENCY, [BASIS])

Returns the first coupon date following the settlement date.

Parameters

```
settlement          Date object: Security's settlement
date
maturity            Date object: Security's maturity date
(date when security expires)
frequency           Number of coupon payments per year
(1=annual, 2=semi-annual, 4=quarterly)
basis               Day count convention code (see below)
```

Optional parameter `basis` indicates the day count convention:

- 0: US 30/360 (default)
- 1: Actual/Actual
- 2: Actual/360
- 3: Actual/365

- 4: European 30/360

Example

First coupon date following settlement date of January 25, 2007, where payments are made twice per year, maturity date is November 15, 2008, and Actual/Actual counting convention is used.

```
CALC.coupncd(new Date(2007,0,25), new Date(2008,10,15), 2,
1);
// returns Tue May 15 00:00:00 EDT 2007
```

With semi-annual payments, the coupon dates in this example are May 15 and November 15, hence the function returns the first payment date of May 15, 2007.

CALC.COUPNUM(SETTLEMENT, MATURITY, FREQUENCY, [BASIS])

Returns the number of coupons payable between the settlement date and maturity date, rounded up to the nearest whole coupon.

Parameters

settlement	Date object: Security's settlement date
maturity	Date object: Security's maturity date (date when security expires)
frequency	Number of coupon payments per year (1=annual, 2=semi-annual, 4=quarterly)
basis	Day count convention code (see below)

Optional parameter `basis` indicates the day count convention:

- 0: US 30/360 (default)
- 1: Actual/Actual
- 2: Actual/360
- 3: Actual/365
- 4: European 30/360

Example

Number of payments between settlement date of January 25, 2007 and maturity date of November 15, 2008, where payments are made twice per year, and Actual/Actual counting convention is used.

```
CALC.coupnum(new Date(2007,0,25), new Date(2008,10,15), 2,
1);
// returns 4
```

With semi-annual payments, the coupon dates in this example are May 15 and November 15, hence there are four payments issued between settlement and maturity.

CALC.COUPPCD(SETTLEMENT, MATURITY, FREQUENCY, [BASIS])

Returns the coupon date that immediately precedes the settlement date.

Parameters

```
settlement          Date object: Security's settlement
date
maturity            Date object: Security's maturity date
(date when security expires)
frequency           Number of coupon payments per year
(1=annual, 2=semi-annual, 4=quarterly)
basis               Day count convention code (see below)
```

Optional parameter `basis` indicates the day count convention:

- 0: US 30/360 (default)
- 1: Actual/Actual
- 2: Actual/360
- 3: Actual/365
- 4: European 30/360

Example

Coupon date preceding a settlement date of January 25, 2007, where maturity date is November 15, 2008, payments are made twice per year, and Actual/Actual counting convention is used.

```
CALC.couppcd(new Date(2007,0,25), new Date(2008,10,15), 2,
1);
// returns Wed Nov 15 00:00:00 EST 2006
```

With semi-annual payments, the coupon dates in this example are May 15 and November 15, hence the coupon date preceding the settlement date is November 15, 2006.

CALC.CUMIPMT(RATE, NPER, PV, START_PERIOD, END_PERIOD, TYPE)

Returns a negative value representing cumulative interest paid on a loan between `start_period` and `end_period`, inclusive.

Parameters

```
rate          Annual interest rate
nper          Total number of monthly periods
pv            Present value of loan
start_period  Integer representing the initial period in the
calculation
end_period    Integer representing the final period in the
calculation
type          Timing of payment
              0: payment at end of period
              1: payment at start of period
```

Example

Cumulative interest paid on a 30-year loan taken at an annual rate of 9% for periods 13 through 24, where each payment is made at the beginning of the period.

```
CALC.cumipmt(0.09/12, 30 * 12, 125000, 13, 24, 1);
// returns -11052.339583871808
```

CALC.CUMPRINC(RATE, NPER, PV, START_PERIOD, END_PERIOD, TYPE)

Returns a negative value representing the cumulative principal paid on a loan between `start_period` and `end_period`, inclusive.

Parameters

rate Annual interest rate
nper Total number of monthly periods
pv Present value of loan
start_period Integer representing the initial period in the calculation
end_period Integer representing the final period in the calculation
type Timing of payment
0: payment at end of period
1: payment at start of period

Example

Cumulative principal paid on a 30-year loan taken at an annual rate of 9% for periods 13 through 24, where each payment is made at the beginning of the period.

```
CALC.cumprinc(0.09/12, 30 * 12, 125000, 13, 24, 1);
// returns -927.1534723780389
```

CALC.DB(COST, SALVAGE, LIFE, PERIOD, [MONTH])

Returns the depreciation of an asset for accounting year `period` using the fixed-declining balance method.

Parameters

cost Initial cost of the asset
salvage Salvage value at end of asset life
life Number of years of depreciation
period Period (year index) for which to return depreciation (in range 1 to life, inclusive)
month Number of months in the first year of depreciation (default=12)

Example

Depreciation on a $1,000,000 asset that has a lifetime of 6 years, and end-of-life salvage value of $100,000, where the first year of depreciation contains 7 months.

```
var deprec = 0;
for(var period = 1; period < 7; period++) {
```

```
  deprec = CALC.db(1000000, 100000, 6, period, 7);
  log('Period ' + period + ': Depreciation of ' + deprec);
}

// Returns:
// Period 1: Depreciation of 186083.33333333334
// Period 2: Depreciation of 259639.41666666666
// Period 3: Depreciation of 176814.44275000002
// Period 4: Depreciation of 120410.63551274998
// Period 5: Depreciation of 81999.64278418274
// Period 6: Depreciation of 55841.75673602846
```

The rate of depreciation is for a given year is

$$rate = \left(\frac{numMonths}{12}\right)\left(1 - \left(\frac{salvage}{\text{cost}}\right)^{1/life}\right)$$

where *numMonths* is the number of months in the year. For the first (fractional) year, *numMonths* is given by parameter `month`. For subsequent full years, *numMonths* is 12.

Depreciation during the final (fractional) period is the difference between the asset's value in the previous period and the asset's salvage value. The function returns 0 for periods after which the asset has already reached its salvage value.

CALC.DDB(COST, SALVAGE, LIFE, PERIOD, [FACTOR])

Returns the depreciation of an asset for accounting year `period` using the double-declining balance method or some other factor.

Parameters

cost Initial cost of the asset
salvage Salvage value at end of asset life
life Number of years of depreciation
period Period (year index) for which to return depreciation (in range 1 to life, inclusive)
factor Rate at which the balance declines (default=2)

Example

Depreciation on a $1,000,000 asset that has a lifetime of 6 years and end-of-life salvage value of $100,000.

```
var deprec = 0;
for(var period = 1; period < 7; period++) {
  deprec = CALC.ddb(1000000, 100000, 6, period);
  log('Period ' + period + ': Depreciation of ' + deprec);
}

// Returns:
// Period 1: Depreciation of 333333.3333333333
// Period 2: Depreciation of 222222.22222222225
// Period 3: Depreciation of 148148.14814814815
// Period 4: Depreciation of 98765.43209876546
// Period 5: Depreciation of 65843.62139917696
// Period 6: Depreciation of 31687.242798353895
```

The rate of depreciation for a given year is `factor/life`. If `factor` is omitted, it is assumed to be 2 (the double-declining balance method).

CALC.DISC(SETTLEMENT, MATURITY, PR, REDEMPTION, [BASIS])

Returns the discount rate for a security.

Parameters

settlement	Date object: Security's settlement date
maturity	Date object: Security's maturity date (date when security expires)
pr	Security's price per $100 face value
redemption	Security's redemption value per $100 face value
basis	Day count convention code (see below)

Optional parameter `basis` indicates the day count convention:

- 0: US 30/360 (default)
- 1: Actual/Actual
- 2: Actual/360
- 3: Actual/365
- 4: European 30/360

Example

Discount rate for a security with price of $97.975, redemption value of $100, settlement date of January 25, 2007, maturity date of June 15, 2007, and Actual/Actual counting convention.

```
CALC.disc(new Date(2007, 0, 25), new Date(2007, 5, 15),
97.975, 100, 1);
// returns 0.0524202127659576 (approx. 5.24%)
```

The rate is calculated as follows:

```
rate = (redemption - pr) / (redemption *
yearfrac(settlement; maturity; basis))
```

CALC.DURATION(SETTLEMENT, MATURITY, RATE, YLD, FREQUENCY, [BASIS])

Returns the Macaulay duration in years for a security with an assumed par value of $100. Duration is the weighted average of the present value of the cash flows and is used as a measure of a bond price's response to changes in yield.

Parameters

settlement	Date object: Security's settlement date
maturity	Date object: Security's maturity date (date when security expires)
rate	Security's annuual coupon rate

yld	Security's annual yeild
frequency	Number of coupon payments per year (1,2, or 4)
basis	Day count convention code (see below)

Optional parameter `basis` indicates the day count convention:

- 0: US 30/360 (default)
- 1: Actual/Actual
- 2: Actual/360
- 3: Actual/365
- 4: European 30/360

Example

Macaulay duration for a security with coupon rate of 8%, annual yield of 9%, settlement date of January 1, 2008, maturity date of January 1, 2016, two coupon payments per year, and Actual/Actual counting convention.

```
CALC.duration(new Date(2008, 0, 1), new Date(2016, 0, 1),
0.08, 0.09, 2, 1);
// returns 5.993774955545185 (years)
```

CALC.EFFECT(NOMINAL_RATE, NPERY)

Returns the effective annual interest rate, given the nominal annual interest rate and the number of compounding periods per year.

Parameters

nominal_rate	Nominal interest rate
npery	number of compounding periods per year

Example

Effective interest rate for a nominal rate of 5.25% with 4 compounding periods per year.

```
CALC.effect(0.0525,4);
// returns 0.05354266737075819 (approximately 5.35%)
```

CALC.FV(RATE, NPER, PMT, [PV], [TYPE])

Returns the future value of an investment based on periodic, constant payments and a constant interest rate.

Parameters

rate	Interest rate per period
nper	Total number of payment periods in an annuity
pmt	constant payment made each period (default=0)
pv	Present value (default=0)
type	Timing of payment 0: payment at end of period (default) 1: payment at start of period

Typically, `pmt` contains principal and interest but no other fees or taxes. If `pmt` is omitted (`null`), you must include the `pv` argument, the lump-sum amount that a series of future payments is worth right now (for a loan, this is the original loan amount). If `pv` is omitted (`null`), it is assumed to be 0 (typical for a bond) and you must include the `pmt` argument.

Example

Future value of a $5000 loan with 6% annual interest rate, 10 payment periods, and $300 payment at start of each period. (Note that payments are entered as negative values.)

```
CALC.fv(0.06/12, 10, -300, 5000, 1);
// returns -2171.9506981443483
```

The result indicates that payment of approximately $4762.75 will be required at the end of the term. If no payments are made (`pmt=0`), then the final payment will be $8954.24, as shown below.

```
CALC.fv(0.06/12, 10, 0, 5000, 1);
// returns -5255.700660203947
```

CALC.FVSCHEDULE(PRINCIPAL, SCHEDULE)

Returns the future value of an initial principal after applying a specified series of compound interest rates. Use this function to calculate the future value of an investment with a variable or adjustable rate

Parameters

principal Present value
schedule Array of interest rates to apply

Example

Future value of a $5000 loan with variable rate of 5%, 6%, and 7% over the three year term.

```
CALC.fvschedule(5000, [0.05,0.06,0.07]);
// returns 5954.55
```

XXXXXXXXX submitted bug

CALC.INTRATE(SETTLEMENT, MATURITY, INVESTMENT, REDEMPTION, [BASIS])

Returns the interest rate for a fully invested security without compounding.

Parameters

settlement Date object: Security’s settlement date
maturity Date object: Security’s maturity date (date when security expires)
investment Amount paid for the security
redemption Security’s redemption value per $100 face value

basis	Day count convention code (see below)

Optional parameter `basis` indicates the day count convention:

- 0: US 30/360 (default)
- 1: Actual/Actual
- 2: Actual/360
- 3: Actual/365
- 4: European 30/360

Example

Effective interest rate without compounding for a security purchased at $100 on January 1, 2008 and redeemed for $200 on January 1, 2016, with Actual/Actual counting convention.

```
CALC.intrate(new Date(2008,0,1), new Date(2016,0,1), 100,
200, 1)
// Returns 0.12525667351129363 (approximately 12.5%)
```

CALC.IPMT(RATE, PER, NPER, PV, FV, TYPE)

Returns the interest in a given period's payment for an investment based on periodic constant payments and a constant interest rate.

Parameters

rate	Interest rate per period
per	Period for which to return interest (between 1 and nper)
nper	Total number of payment periods in an annuity
pv	Present value (positive for a loan, negative for an investment)
fv	Desired cash balance after last payment
type	Timing of payment 0: payment at end of period (default) 1: payment at start of period

The present value, `pv`, is the lump-sum amount that a series of future payments is worth right now. The future value, `fv`, is a cash balance you want to attain after the last payment is made (generally 0 for a loan). Use *CALC.ppmt(rate, per, nper, pv, fv, type)* to calculate the portion of capital in a given payment. Use *CALC.pmt(rate, nper, pv, fv, type)* to calculate the total payment (capital and interest).

Example

Amount of interest in each of the first ten payments for an $8000 two-year loan with 10% annual interest and payment at end of period. (The negative values indicates that these are payments.)

```
for(var period=1; period<=10; period++){
  log('Period ' + period +': ' + CALC.ipmt(0.1/12, period,
12*2, 8000, 0, 0))
}
```

```
// Returns:
// Period 1: -66.66666666666667
// Period 2: -64.14589379972111
// Period 3: -61.60411449221768
// Period 4: -59.04115369048507
// Period 5: -56.45683488207135
// Period 6: -53.85098008358753
// Period 7: -51.223409828449675
// Period 8: -48.57394315451896
// Period 9: -45.902397591638874
// Period 10: -43.20858914906812
```

CALC.ISPMT(RATE, PER, NPER, PV)

Returns the interest paid during a given period for a fixed-rate investment in which a constant amount of the principal is paid at the start of each period. This function is provided for compatibility with Lotus 1-2-3.

Parameters

rate Interest rate per period
per Period for which to return interest (between 1 and nper)
nper Total number of payment periods in an annuity
pv Present value (positive for a loan, negative for an investment)

Example

Amount of interest in each of the first ten payments for an $8000 two-year loan with 10% annual interest. (The negative values indicates that these are payments.)

```
for(var period=1; period<=10; period++){
  log('Period ' + period +': ' + CALC.ispmt(0.1/12, period,
12*2, 8000))
}

// Returns:
// Period 1: -63.88888888888889
// Period 2: -61.111111111111114
// Period 3: -58.333333333333336
// Period 4: -55.555555555555564
// Period 5: -52.77777777777778
// Period 6: -50
// Period 7: -47.22222222222223
// Period 8: -44.44444444444445
// Period 9: -41.66666666666667
// Period 10: -38.88888888888889
```

CALC.MDURATION(SETTLEMENT, MATURITY, RATE, YLD, FREQUENCY, [BASIS])

Macaulay duration is given by *CALC.duration(settlement, maturity, rate, yld, frequency, [basis])*.

Returns the modified Macaulay duration (in years) for a security with an assumed par value of $100. This is the Macaulay duration divided by (`1 + yld/frequency`).

Parameters

settlement Date object: Security’s settlement date

maturity Date object: Security's maturity date (date when security expires)
rate Security's annual coupon rate
yld Security's annual yeild
frequency Number of coupon payments per year (1,2, or 4)
basis Day count convention code (see below)

Optional parameter `basis` indicates the day count convention:

- 0: US 30/360 (default)
- 1: Actual/Actual
- 2: Actual/360
- 3: Actual/365
- 4: European 30/360

Example

Modified Macaulay duration for a security with coupon rate of 8%, annual yield of 9%, settlement date of January 1, 2008, maturity date of January 1, 2016, two coupon payments per year, and Actual/Actual counting convention.

```
CALC.mduration(new Date(2008, 0, 1), new Date(2016, 0, 1),
0.08, 0.09, 2, 1);
// returns 5.735669813918838 (years)
```

CALC.MIRR(VALUES, FINANCE_RATE, REINVEST_RATE)

Returns the modified internal rate of return for a series of periodic cash flows. This function considers both the cost of the investment and the interest received on reinvestment of cash.

Parameters

values Array of numbers representing a series of payments (negative values) and income (positive values) occurring at regular periods.
finance_rate Interest rate payed on money used in the cash flows
reinvest_rate Interest rate on reinvested cash flows

The `values` array must contain at least one negative and one positive value. Typically, for an investment, the first value is the (negative) amount of the investment, and subsequent values are (positive) payments which are reinvested at the `reinvest_rate`.

Example

Modified interest rate for an investment of $120,000 at a 10% interest rate, where payments of $39,000, $30,000, $21,000, $37,000, and $46,000 are reinvested at 12%.

```
CALC.mirr([-120000, 39000, 30000, 21000, 37000, 46000], 0.1,
0.12);
// returns 0.1260941303659051 (approximately 12.6%)
```

CALC.NOMINAL(EFFECT_RATE, NPERY)

Returns the nominal annual interest rate, given the effective compounded rate and the number of compounding periods per year.

Parameters

effect_rate	Efective interest rate
nper	Number of compounding periods per year

Example

Nominal rate for an investment with an effective rate of 5.3543%, where interest is compounded four times per year.

Example

```
CALC.nominal(0.053543, 4);
// returns 0.052500319868356016 (approximately 5.25%)
```

CALC.NPER(RATE, PMT, PV, FV, TYPE)

Returns the number of periods for an investment based on periodic, constant payments and a constant interest rate.

Parameters

rate	Interest rate per period
pmt	Payment made each period; constant over life of annuity
pv	Present value (positive for a loan, negative for an investment)
fv	Desired cash balance after last payment
type	Timing of payment 0: payment at end of period (default) 1: payment at start of period

Typically, `pmt` contains principal and interest but no other fees or taxes. The present value, `pv`, is the lump-sum amount that a series of future payments is worth right now. The future value, `fv`, is a cash balance you want to attain after the last payment is made (generally 0 for a loan).

Example

Number of periods for an initial $1000 investment to grow to a future value of $10,000 with 12% interest per period (compounded) and payments of $100 at the start of each period. (Negative values indicate payments.)

```
CALC.nper(0.12, -100, -1000, 10000, 1);
// returns 15.288014619990111 (approximately 15 periods)
```

CALC.NPV(RATE, VALUES)

Returns the net present value of an investment by using a discount rate and a series of future payments (negative values) and income (positive values).

Parameters

rate Discount rate over a period
values Array of payment and income values

Example

Net present value of an investment, under an assumed discount rate of 10%, that has first period payment (investment) of $10,000, second period income of $3000, third period income of $4200, and fourth period income of $6800.

```
CALC.npv(0.10, [-10000, 3000, 4200, 6800]);
// returns 1188.4434123352207 ($1,188.44 present value)
```

CALC.PMT(RATE, NPER, PV, FV, TYPE)

Returns the periodic payment for a loan based on constant payments and a constant interest rate.

Parameters

rate Interest rate per period
nper Total number of monthly payments for the loan
pv Present value (positive for a loan, negative for an investment)
fv Desired cash balance after last payment
type Timing of payment
0: payment at end of period (default)
1: payment at start of period

Typically, `pmt` contains principal and interest but no other fees or taxes. The present value, `pv`, is the lump-sum amount that a series of future payments is worth right now. The future value, `fv`, is a cash balance you want to attain after the last payment is made (generally 0 for a loan). Use *CALC.ppmt(rate, per, nper, pv, fv, type)* to calculate the portion of capital in a given payment. Use *CALC.ipmt(rate, per, nper, pv, fv, type)* to calculate the portion of interest in a given payment.

Example

Payment at start of each period for a $10,000 2-year loan with 8% interest per period. (Negative values indicate payments.)

```
CALC.pmt(0.08/12, 24, 10000, 0, 1);
// returns -449.27772969719734 (approximately $449.28 per
period)
```

CALC.PPMT(RATE, PER, NPER, PV, FV, TYPE)

Returns the capital in a given period's payment for an investment based on periodic constant payments and a constant interest rate.

Parameters

rate Interest rate per period
per Period for which to return capital repayment (between 1 and nper)
nper Total number of payment periods in an annuity
pv Present value (positive for a loan, negative for an investment)
fv Desired cash balance after last payment
type Timing of payment
0: payment at end of period (default)
1: payment at start of period

The present value, `pv`, is the lump-sum amount that a series of future payments is worth right now. The future value, `fv`, is a cash balance you want to attain after the last payment is made (generally 0 for a loan). Use *CALC.ipmt(rate, per, nper, pv, fv, type)* to calculate the portion of interest in a given payment. Use *CALC.pmt(rate, nper, pv, fv, type)* to calculate the total payment (capital and interest).

Example

Amount of capital repaid in each of the first ten payments for an $8000 two-year loan with 10% annual interest and payment at end of period. (The negative values indicates that these are payments.)

```
for(var period=1; period<=10; period++){
  log('Period ' + period +': ' + CALC.ppmt(0.10/12, period,
12*2, 8000, 0, 0))
}

// Returns:
// Period 1: -302.4927440334672
// Period 2: -305.0135169004128
// Period 3: -307.55529620791623
// Period 4: -310.1182570096488
// Period 5: -312.70257581806254
// Period 6: -315.3084306165464
// Period 7: -317.9360008716842
// Period 8: -320.58546754561496
// Period 9: -323.257013108495
// Period 10: -325.95082155106576
```

CALC.PRICE(SETTLEMENT, MATURITY, RATE, YLD, REDEMPTION, FREQUENCY, [BASIS])

Returns the clean price per $100 face value of a security that pays periodic interest.

Parameters

settlement Date object: Security's settlement date

maturity	Date object: Security's maturity date (date when security expires)
rate	Security's annual coupon rate
yld	Security's annual yeild
redemption	Security's redemption value per $100 face value
frequency	Number of coupon payments per year (1,2, or 4)
basis	Day count convention code (see below)

Optional parameter `basis` indicates the day count convention:

- 0: US 30/360 (default)
- 1: Actual/Actual
- 2: Actual/360
- 3: Actual/365
- 4: European 30/360

Example

Clean price for a security with coupon rate of 5.75%, annual yield of 6.5%, redemption price of $100, settlement date of February 15, 2008, maturity date of October 15, 2017, two coupon payments per year, and US 30/360 counting convention.

```
CALC.price(new Date(2008,1,15), new Date(2017,10,15),
0.0575, 0.065, 100, 2, 0);
// returns 94.63436162132213 (approximately $94.63)
```

CALC.PRICEDISC(SETTLEMENT, MATURITY, DISCOUNT, REDEMPTION, [BASIS])

Returns the price per $100 face value of a discounted security (zero-coupon bond).

Parameters

settlement	Date object: Security's settlement date
maturity	Date object: Security's maturity date (date when security expires)
discount	Security's discount rate
redemption	Security's redemption value per $100 face value
basis	Day count convention code (see below)

Optional parameter `basis` indicates the day count convention:

- 0: US 30/360 (default)
- 1: Actual/Actual
- 2: Actual/360
- 3: Actual/365
- 4: European 30/360

Example

Price for a bond with par value of $100, discount rate of 5.75%, settlement date of February 15, 2008, maturity date of October 15, 2017, and US 30/360 counting convention.

```
CALC.pricedisc(new Date(2008,1,15), new Date(2017,10,15),
0.0575, 100, 0);
// returns 43.9375 (approximately $43.94)
```

CALC.PRICEMAT(SETTLEMENT, MATURITY, ISSUE, RATE, YLD, [BASIS])

Returns the price per $100 face value of a security that pays interest at maturity.

Parameters

settlement	Date object: Security's settlement date
maturity	Date object: Security's maturity date (date when security expires)
issue	Date object: Security's issue date
rate	Security's annual coupon rate (paid at maturity)
yld	Security's annual yeild
basis	Day count convention code (see below)

Optional parameter `basis` indicates the day count convention:

- 0: US 30/360 (default)
- 1: Actual/Actual
- 2: Actual/360
- 3: Actual/365
- 4: European 30/360

Example

Price for a security with coupon rate of 6.1%, annual yield of 6.1%, settlement date of February 15, 2008, maturity date of April 13, 2008, issue date of November 11, 2007, and US 30/360 counting convention.

```
CALC.pricemat(new Date(2008,1,15), new Date(2008,3,13), new
Date(2007, 10, 11), 0.061, 0.061, 0);
// returns 99.98449887555694 (approximately $99.98)
```

CALC.PV(RATE, NPER, PMT, FV, TYPE)

Returns the present value of an investment. The present value, `pv`, is the lump-sum amount that a series of future payments is worth right now.

Parameters

rate	Interest rate per period
nper	Total number of payment periods in an annuity
pmt	Fixed payment made each period
fv	Desired cash balance after last payment

type	Timing of payment 0: payment at end of period (default) 1: payment at start of period

Typically, `pmt` includes principal and interest but no other fees or taxes. The future value, `fv`, is a cash balance you want to attain after the last payment is made (generally 0 for a loan).

Example

Present value of a 20-year loan with interest rate of 8% and payments of $500 at the end of each period. (Payments are negative, income is positive.)

```
CALC.pv(0.08/12, 20*12, -500, 0, 0);
// returns 59777.145851187764 (aproximately $59,777.15)
```

CALC.RECEIVED(SETTLEMENT, MATURITY, INVESTMENT, DISCOUNT, [BASIS])

Returns the amount received at maturity for a fully invested security with no interest payments.

Parameters

settlement	Date object: Security's settlement date
maturity	Date object: Security's maturity date (date when security expires)
investment	Amount invested in the security
discount	Security's discount rate
basis	Day count convention code (see below)

Optional parameter `basis` indicates the day count convention:

- 0: US 30/360 (default)
- 1: Actual/Actual
- 2: Actual/360
- 3: Actual/365
- 4: European 30/360

Example

Amount received at maturity for a $1,000,000 investment made at 5.75%, with settlement date of February 15, 2008, maturity date of May 15, 2008, and Actual/360 counting convention.

```
CALC.received(new Date(2008,1,15), new Date(2008,4,15),
1000000, 0.0575, 2);
// returns 1014584.6544071021 (approximately $1,014,584.65)
```

CALC.SLN(COST, SALVAGE, LIFE)

Returns the straight-line depreciation of an asset for one period (year).

Parameters

cost Initial cost of the asset
salvage Salvage value at end of asset life
life Number of years of depreciation

Example

Depreciation on a $1,000,000 asset that has a lifetime of 6 years and end-of-life salvage value of $100,000.

```
CALC.sln(1000000, 100000, 6);
// Returns 150000.0 ($150,000 depreciation per year)
```

CALC.SYD(COST, SALVAGE, LIFE, PER)

Returns the sum-of-years' digits depreciation of an asset for a specified period.

Parameters

cost Initial cost of the asset
salvage Salvage value at end of asset life
life Number of years of depreciation
per Period for which to return depreciation (in range 1 to life, inclusive)

Example

Depreciation on a $30,000 asset that has a lifetime of 10 years and end-of-life salvage value of $7,500.

```
for(var period=1; period<=10; period++){
  log('Period ' + period +': ' + CALC.syd(30000, 7500, 10, period))
}

// Returns:
// Period 1: 4090.909090909091
// Period 2: 3681.818181818182
// Period 3: 3272.7272727272725
// Period 4: 2863.6363636363635
// Period 5: 2454.5454545454545
// Period 6: 2045.4545454545455
// Period 7: 1636.3636363636363
// Period 8: 1227.2727272727273
// Period 9: 818.1818181818181
// Period 10: 409.09090909090907
```

CALC.TBILLEQ(SETTLEMENT, MATURITY, DISCOUNT)

Returns the bond-equivalent yield (BEY) for a Treasury bill using the 360-day year convention.

Parameters

settlement Date object: T-bill's settlement date
maturity Date object: T-bill's maturity date
discount T-bill's discount rate

Example

Required bond yield (payable at maturity) to match a Treasury bill with 9.14% discount rate, settlement date of March 31, 2008, and maturity date of June 1, 2008.

```
CALC.tbilleq(new Date(2008, 2, 31), new Date(2008, 5,1),
0.0914);
// returns 0.09415149356594303 (approximately 9.42%)
```

CALC.TBILLPRICE(SETTLEMENT, MATURITY, DISCOUNT)

Returns the issue price per $100 face value for a Treasury bill using the 360-day year convention. The **`settlement`** and `maturity` dates must be within one calendar year of each other.

Parameters

`settlement`	Date object: T-bill's settlement date
`maturity`	Date object: T-bill's maturity date
`discount`	T-bill's discount rate

Example

Issue price of a Treasury bill with 9.14% discount rate, settlement date of March 31, 2008, and maturity date of June 1, 2008.

```
CALC.tbillprice(new Date(2008, 2, 31), new Date(2008, 5,1),
0.0914);
// returns 98.42588888888889 (approximately $98.43 per $100
face value)
```

CALC.TBILLYIELD(SETTLEMENT, MATURITY, PR)

Returns the yield for a Treasury bill using the 360-day year convention. The **`settlement`** and `maturity` dates must be within one calendar year of each other.

Parameters

`settlement`	Date object: T-bill's settlement date
`maturity`	Date object: T-bill's maturity date
`pr`	T-bill's issue price per $100 face value

Example

Yield for a Treasury bill purchased at $98.45 per $100 face value, settlement date of March 31, 2008, and maturity date of June 1, 2008.

Example

```
CALC.tbillyield(new Date(2008, 2, 31), new Date(2008, 5,1),
98.45);
// returns 0.09141696292534264 (approximately 9.14%)
```

CALC.VDB(COST, SALVAGE, LIFE, START_PERIOD, END_PERIOD, FACTOR, NO_SWITCH)

Returns the depreciation of an asset over an arbitrary period using the variable-declining balance method.

Parameters

cost Initial cost of the asset
salvage Salvage value at end of asset life
life Number of years of depreciation
start_period Real number representing the initial period in the calculation (in range 0 to end_period, inclusive)
end_period Real number representing the final period in the calculation (in range start_period to life, inclusive)
factor Rate at which the balance declines (default=2)
no_switch true: always use variable-declining balance calculation
false: switch to straight-line depreciation when result is greater than declining-balance calculation (default)

The rate of depreciation for a given year is `factor/life`. If `factor` is omitted, it is assumed to be 2 (the double-declining balance method).

Example

Depreciation over the first 5 years (period 0 to period 5) for an asset purchased at $2400, with salvage value of $300 after 10 years, and depreciation factor of 1.5.

```
CALC.vdb(2400, 300, 10, 0, 5, 1.5);
// returns 975.1072500000002 (approximately $975.11)
```

CALC.XNPV(RATE, VALUES, DATES)

Returns the net present value for a schedule of cash flows that is not necessarily periodic, based on a 365-day year.

Parameters

rate Discount rate to apply to the cash flows
values Array of numbers representing a series of payments (negative values) and income (positive values) occurring at the specified dates
dates Array of dates on which payments are made (first value must be the earliest date)

The array of `values` must contain at least one positive value and one negative value. Typically, for an investment, the first value is the (negative) amount of the investment, and subsequent values are (positive) payments. The `dates` array can list the dates in any order so long as the first item is the earliest date in the series.

Example

Net present value for a $10,000 investment made on January 1, 2008 at a 9% interest rate, where the following subsequent payment are made:

– $2,750 on March 1, 2008
– $4,250 on October 30, 2008
– $3,250 on February 15, 2009
– $2,750 on April 1, 2009

```
CALC.xnpv(0.09, [-10000, 2750, 4250, 3250, 2750],[new
Date(2008, 0, 1),new Date(2008, 2, 1),new Date(2008, 9,
30),new Date(2009, 1, 15), new Date(2009, 3, 1)]);
// returns 2086.647602031535 (approximately $2,086.65)
```

CALC.YIELDDISC(SETTLEMENT, MATURITY, PR, REDEMPTION, [BASIS])

Returns the annual yield for a discounted security that does not pay interest.

Parameters

settlement	Date object: Security's settlement date
maturity	Date object: Security's maturity date (date when security expires)
pr	Security's price per $100 face value
redemption	Security's redemption value per $100 face value
basis	Day count convention code (see below)

Optional parameter `basis` indicates the day count convention:

- 0: US 30/360 (default)
- 1: Actual/Actual
- 2: Actual/360
- 3: Actual/365
- 4: European 30/360

Example

Yield for a bond with price of $99.79, redemption value of $100, settlement date of February 16, 2008, maturity date of March 1, 2008, and Actual/360 counting convention.

```
CALC.yielddisc(new Date(2008, 1, 16), new Date(2008, 2,1),
99.79, 100, 2);
// returns 0.054113638641144796 (approximately 5.41%)
```

CALC.YIELDMAT(SETTLEMENT, MATURITY, ISSUE, RATE, PR, [BASIS])

Returns the annual yield of a security that pays interest only at maturity.

Parameters

settlement	Date object: Security's settlement date

maturity	Date object: Security's maturity date (date when security expires)
issue	Date object: Security's issue date
rate	Security's interest rate at date of issue
pr	Security's price per $100 face value
basis	Day count convention code (see below)

Optional parameter `basis` indicates the day count convention:

- 0: US 30/360 (default)
- 1: Actual/Actual
- 2: Actual/360
- 3: Actual/365
- 4: European 30/360

Example

Yield for a 6.25% bond with price of $100.00, issue date of November 8, 2007, settlement date of March 15, 2008, maturity date of November 3, 2008, and US 30/360 counting convention.

Example

```
CALC.yieldmat(new Date(2008, 2, 15), new Date(2008, 10,3),
new Date(2007, 10,8), 0.0625, 100.00, 0);
// returns 0.06115169016476979 (approximately 6.12%)
```

JS.7.3 Logical Functions

The CALC object provides the logical functions listed below.

CALC.AND(BOOLEAN_1, BOOLEAN_2, …, BOOLEAN_N)

Returns `true` only if all arguments evaluate to `true`, otherwise returns `false`.

Parameters

boolean_1	Boolean, or expression that evaluates to Boolean
boolean_2	Boolean, or expression that evaluates to Boolean
boolean_n	Boolean, or expression that evaluates to Boolean

Example

```
CALC.and((0<3),(-1>2),(5<6))
// returns false
```

CALC.IIF(CONDITION, VALUE_IF_TRUE, VALUE_IF_FALSE)

Returns `value_if_true` if `condition` evaluates to `true`, and returns `value_if_false` if `condition` evaluates to `false`. This is similar to the JavaScript ternary operator, which can also be used for this purpose.

Parameters

```
condition     Logical condition to test
value_if_true Expression to evaluate if condition is true
value_if_falseExpression to evaluate if condition is false
```

Example

```
for(var value=-2; value<=2; value++) {
  log(CALC.iif(value < 0, "less than 0", "greater than or
equal to 0"))
}

// Returns:
// less than 0
// less than 0
// greater than or equal to 0
// greater than or equal to 0
// greater than or equal to 0
```

CALC.NOT(BOOLEAN)

Reverses the value of the Boolean argument. This is similar to the JavaScript "`!`" operator, which can also be used for this purpose.

Parameters

```
boolean       Boolean value to reverse
```

Example

```
CALC.not(0<3)
// returns false
```

CALC.OR(BOOLEAN_1, BOOLEAN_2, ..., BOOLEAN_N)

Returns `true` if one or more of the arguments evaluate to `true`, otherwise returns `false`.

Parameters

```
boolean_1     Boolean, or expression that evaluates to
Boolean
boolean_2     Boolean, or expression that evaluates to
Boolean
boolean_n     Boolean, or expression that evaluates to
Boolean
```

Example

```
CALC.or((0<3),(-1>2),(5<6))
// returns true
```

JS.7.4 Math Functions

This section presents the math functions provided by the CALC object.

CALC.ABS(NUMBER)

Returns the absolute value of a number.

Parameters

number A numerical value

Example

```
CALC.abs(-6)
// returns 6.0
```

CALC.ACOS(NUMBER)

Returns the inverse cosine of a number in radians.

Parameters

number A numerical value between -1 and 1

Example

```
CALC.acos(-1)
// returns 3.141592653589793 (pi radians)
```

CALC.ACOSH(NUMBER)

Returns the inverse hyperbolic cosine of a number.

Parameters

number A numerical value greater than or equal to 1

Example

```
CALC.acosh(10)
// returns 2.993222846126381
```

CALC.ASIN(NUMBER)

Returns the inverse sine of a number in radians.

Parameters

number A numerical value between -1 and 1

Example

```
CALC.asin(1);
// returns 1.5707963267948966 (pi/2 radians)
```

CALC.ASINH(NUMBER)

Returns the inverse hyperbolic sine of an number.

Parameters

number A numerical value greater than or equal to 1

Example

```
CALC.asinh(77);
// returns 5.03699476537355
```

CALC.ATAN(NUMBER)

Returns the inverse tangent of a number in radians.

Parameters

number A numerical value

Example

```
CALC.atan(1);
// returns 0.7853981633974475 (pi/4 radians)
```

CALC.ATAN2(NUMBERX, NUMBERY)

Returns the inverse tangent of the point represented by the specified *x* and *y* co-ordinates, in radians.

Parameters

numberX X coordinate
numberY Y coordinate

Example

```
CALC.atan2(1,1);
// 0.7853981633974475 (pi/4 radians)
```

CALC.ATANH(NUMBER)

Return the inverse hyperbolic tangent of a number in radians.

Parameters

number A numerical value between -1 and 1 (exclusive)

Example

```
CALC.atanh(0.5);
// returns 0.5493061443340549
```

CALC.CEILING(NUMBER, SIGNIFICANCE)

Returns `number` rounded away from zero to the nearest multiple of `significance`. Positive numbers are rounded up, negative numbers are rounded down.

Parameters

number Number to round
significance Rounding interval (must have same sign as number)

Example

```
CALC.ceiling(4.43, 0.05);
// returns 4.45
```

CALC.COMBIN(TOTALNUMBER, GROUPNUMBER)

Returns the number of unique combinations in which a set of elements can be combined into groups of a given size.

Parameters

totalNumber	Total number of elements available to combine
groupNumber	Size of groups into which elements are combined

Example

Number of unique ways that 10 items can be combined into groups of two:

```
CALC.combin(10, 2);
// returns 45.0
```

CALC.COS(NUMBER)

Returns the cosine of a number representing an angle in radians.

Parameters

number	An angle in radians

Example

```
CALC.cos(pi())
// returns -1.0
```

CALC.COSH(NUMBER)

Return the hyperbolic cosine of any real number.

Parameters

number	A real number

Example

```
CALC.cosh(2.7)
// returns 7.473468618806294
```

CALC.DEGREES(NUMBER)

Converts a number representing an angle in radians to degrees.

Parameters

number	An angle in radians

Example

```
CALC.degrees(pi()/2);
// returns 89.99999999999991 (approximately 90 degrees)
```

CALC.EVEN(NUMBER)

Rounds a number away from 0 to the nearest even integer. Positive values are rounded up, negative values are rounded down.

Parameters

number A number to round

Example

```
CALC.even(14.1);
// returns 16
```

CALC.EXP(NUMBER)

Returns the value of the natural exponential function, e^x, where *e* is the base of the natural logarithm and *x* is a specified number.

Parameters

number Exponent

Example

```
CALC.exp(3.2);
// returns 24.532530197109352
```

CALC.FACT(NUMBER)

Returns the factorial of a number. (The factorial of 0 is 1 by definition.)

Parameters

number Non-negative value (n) for which to compute factorial (n!). Any fractional component is ignored.

Example

```
CALC.fact(5);
// returns 120.0 (5x4x3x2x1)
```

CALC.FACTDOUBLE(NUMBER)

Returns the double factorial of a number, *n*.

- If *n* is even, *n*!! = *n*(*n*-2)(*n*-4)...(4)(2). The double factorial of 0 is 1.
- If *n* is odd, *n*!! = *n*(*n*-2)(*n*-4)...(3)(1)

Parameters

number Non-negative value (n) for which to compute double factorial (n!!). Any fractional component is ignored.

Example

```
CALC.factdouble(6);
// returns 48.0
```

CALC.FLOOR(NUMBER, SIGNIFICANCE)

Returns `number` rounded toward zero to the nearest multiple of `significance`. Positive numbers are rounded down, negative numbers are rounded up.

Parameters

number Number to round
significance Rounding interval (must have same sign as number)

Example

```
CALC.floor(4.43, 0.05);
// returns 4.4
```

CALC.GCD(ARRAY)

Returns the greatest common divisor of an array of non-negative numbers. Non-integer values are truncated.

Parameters

array Array of non-negative numbers for which to find GCD

Example

```
CALC.gcd([24, 32, 16, 12]);
// returns 4
```

CALC.INTEGER(NUMBER)

Rounds `number` down to the nearest integer.

Parameters

number Number to round

Example

```
CALC.integer(15.9);
// returns 15
```

CALC.LCM(ARRAY)

Returns the least common multiple of an array of non-negative numbers. Non-integer values are truncated.

Parameters

array Array of positive integers for which to find LCM

Example

```
CALC.lcm([24, 32, 16, 12]);
// returns 96
```

CALC.LN(NUMBER)

Returns the natural logarithm of a positive number (base *e*).

Parameters

number Positive real number

Example

```
CALC.ln(Math.E);
// returns 1.0
```

CALC.LOG(NUMBER, BASE)

Returns the logarithm of a positive number to the specified base.

Parameters

number	Positive number
base	Base of the logarithm, a positive real number

Example

```
CALC.log(64, 4);
// returns 3.0
```

CALC.LOG10(NUMBER)

Returns the logarithm of a positive number to the base 10.

Parameters

number	Positive real number

Example

```
CALC.log10(100);
// returns 2.0
```

CALC.MDETERM(ARRAY)

Returns the matrix determinant of the specified square matrix (two-dimensional array).

Parameters

array	A 2D array representing a square matrix

Example

```
CALC.mdeterm([[1,0,0],[0,1,0],[0,0,1]]);
// returns 1.0
```

CALC.MINVERSE(ARRAY)

Returns the inverse of a nonsingular square matrix (two-dimensional array).

Parameters

array	A 2D array representing a nonsingular square matrix

Example

```
var inv = CALC.minverse([[1,0,1],[0,1,1],[0,0,1]]);
for (var i=0;i<=2;i++){
  log(inv[i].join(","))
}
```

```
// Returns:
// 1,0,-1
// 0,1,-1
// 0,0,1
```

CALC.MMULT(ARRAY1, ARRAY2)

Returns the product of two conformable matrices, `array1*array2`, each represented as a two-dimensional array.

Parameters

array1	A 2D array representing the left matrix
array2	A 2D array representing the right matrix

Example

```
var matrixA = [[1,2],[3,4]];
var matrixB = [[5,6,7],[8,9,0]];
var prod = CALC.mmult(matrixA, matrixB);
for (var i=0;i<=1;i++){
  log(prod[i].join(","))
}

// Returns:
// 21,24,7
// 47,54,21
```

CALC.MOD(NUMBER, DIVISOR)

Returns the remainder when `number` is divided by `divisor`.

Parameters

number	Real number to be divided
divisor	Real number by which to divide

Example

```
log(CALC.mod(3, 2))
// Returns 1.0
```

CALC.MROUND(NUMBER, FACTOR)

Returns `number` rounded to the nearest multiple of `factor`.

Parameters

number	Number to round
factor	Number multiplied to obtain rounding increments

Example

```
CALC.mround(10, 3);
// returns 9

CALC.mround(10.1, 3.3);
// returns 9.899999999999999
```

CALC.MULTINOMIAL(ARRAY)

Returns the ratio of the factorial of a sum of values to the product of their factorials. For three numbers *a*, *b*, and *c* the multinomial is given by

$$\frac{(a+b+c)!}{a!b!c!}$$

Non-integer values are truncated.

Parameters

array Array of positive integers for which to compute multinomial

Example

```
CALC.multinomial([2,3,4]);
// returns 1260.0
```

CALC.ODD(NUMBER)

Rounds a number away from 0 to the nearest odd integer. Positive values are rounded up, negative values are rounded down.

Parameters

number A number to round

Example

```
CALC.odd(13.1);
// returns 15
```

CALC.PI()

Returns the number π, accurate to 15 digits.

Example

```
CALC.pi();
// returns 3.14159265358979
```

CALC.POWER(NUMBER, POWER)

Returns the result of a number raised to a power, b^n.

Parameters

number The base b
power The exponent n

Example

```
CALC.power(2,3);
// returns 8.0
```

CALC.PRODUCT(ARRAY)

Returns the product of all the elements in an array.

Parameters

array Array of elements to multiply

Example

```
CALC.product([2,3,4,5]);
// returns 120.0
```

CALC.QUOTIENT(NUMERATOR, DENOMINATOR)

Returns the integer portion of the quotient `numerator/denominator`.

Parameters

numerator The numerator
denominator The non-zero denominator

Example

```
CALC.quotient(5,2);
// returns 2
```

CALC.RADIANS(NUMBER)

Converts a number representing an angle in degrees to radians.

Parameters

number An angle in degrees

Example

```
CALC.radians(90);
// returns 1.5707963267948966 (π/2 radians)
```

CALC.ROMAN(NUMBER)

Converts an Arabic numeral to Roman numeral, as text.

Parameters

number An number in Arabic (standard) numerals

Example

```
CALC.roman(77);
// returns LXXVII
```

CALC.ROUND(NUMBER, NUM_DIGITS)

Rounds a number to a specified number of digits.

Parameters

number Number to round
num_digits Number of digits (to right of decimal point) to preserve after rounding

Example

```
CALC.round(2.15, 1);
// returns 2.2
```

CALC.ROUNDDOWN(NUMBER, NUM_DIGITS)

Rounds a number toward zero to a specified number of digits. Positive number are rounded down, negative numbers are rounded up.

Parameters

number Number to round
num_digits Number of digits (to right of decimal point) to preserve after rounding

Example

```
CALC.rounddown(3.14159, 3);
// returns 3.141
```

CALC.ROUNDUP(NUMBER, NUM_DIGITS)

Rounds a number away from zero to a specified number of digits. Positive number are rounded up, negative numbers are rounded down.

Parameters

number Number to round
num_digits Number of digits (to right of decimal point) to preserve after rounding

Example

```
CALC.roundup(3.14159, 3);
// returns 3.142
```

CALC.SERIESSUM(X, N, M, COEFFICIENTS)

Returns the following power series sum:

$$a_0x^n + a_1x^{(n+m)} + a_2x^{(n+2m)} + \ldots + a_ix^{(n+im)}$$

Parameters

x The base x
n **Integer e**xponent in first term
m Integer interval by which to increase exponent n in subsequent terms
coefficients Array of coefficients [a0, a1, a2, ..., ai]

Example

```
CALC.seriessum(2, 0, 1, [1,2,3]);
// returns 17.0
```

CALC.SIN(NUMBER)

Returns the sine of the given angle specified in radians.

Parameters

number An angle in radians

Example

```
CALC.sin(pi())
// returns 3.2310891488651735E-15 (approximately 0)
```

CALC.SINH(NUMBER)

Returns the hyperbolic sine of any real number.

Parameters

number A real number

Example

```
CALC.sinh(5.5)
// returns 122.34392274639096
```

CALC.SQRT(NUMBER)

Returns the square root of a positive number.

Parameters

number A positive number

Example

```
CALC.sqrt(5.5)
// returns 2.345207879911715
```

CALC.SQRTPI(NUMBER)

Returns the square root of a positive number multiplied by π, `sqrt(number*pi())`.

Parameters

number A positive number

Example

```
CALC.sqrtpi(5.5)
// returns 4.156772737923478
```

CALC.SUBTOTAL(FUNCTIONNUMBER, ARRAY)

Returns the subtotal of an array based on the specified aggregation function.

Parameters

functionNumber Aggregation function
1: Average of items
2: Count of items
3: Count of non-null items
4: Maximum value of items
5: Minimum value of items

```
                    6: Product of items
                    7: Sample standard deviation of items
                    8: Population standard deviation of
items
                    9: Sum of items
                    10: Sample varianve of items
                    11: Poulation variance of items
array               Array of numbers
```

Example

```
CALC.subtotal(1, [1,2,3]);
// returns 2.0
```

CALC.SUM(ARRAY)

Returns the sum of all the elements in an array.

Parameters

```
array         Array of elements to sum
```

Example

```
CALC.sum([2,3,4,5]);
// returns 14.0
```

CALC.SUMIF(ARRAY1, CONDITIONSTRING, ARRAY2)

Returns the sum of the elements in `array2` corresponding to elements in `array1` that satisfy the condition given by `conditionString`. Examples of valid condition strings are `'>30000'`, `'=4000'`, and `'<3500'`.

Parameters

```
array1              Array of elements to test against
condition
conditionString     String: Condition to test
array2              Array containing values to sum
```

Example

Sum the values in array `[1,2,3,4,5]` having same positions as the values in array `[10,25,30,15,45]` that are greater than 20.

```
CALC.sumif([10,25,30,15,45], '> 20',[1,2,3,4,5]);
// returns 10.0 (i.e., 2+3+5)
```

CALC.SUMPRODUCT(2DARRAY)

Multiplies corresponding row components of a two-dimensional array (matrix) and returns the sum of those products. For example, for a matrix

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

this returns the result *adg+beh+cfi*.

Parameters

2Darray 2D array of values

Example

```
CALC.sumproduct([[1,2,3],[2,1,2],[3,3,1]]);
// returns 18.0
```

CALC.SUMSQ(ARRAY)

Returns the sum of the squares of all the elements in the array.

Parameters

array Array of values

Example

```
CALC.sumsq([1,2,3]);
// returns 14.0
```

CALC.SUMX2MY2(ARRAY1, ARRAY2)

Returns the sum of the *difference* of squares of corresponding values in two arrays. For arrays `[a1,a2,a3,...,an]` and `[b1,b2,b3,...,bn]`, the result gives

$$(a1^2 - b1^2) + (a2^2 - b2^2) + (a3^2 - b3^2) + \ldots + (an^2 - bn^2)$$

Parameters

array1 An array of values [a1,a2,...,an]
array2 An array of values [b1,b2,...,bn]

Example

```
CALC.sumx2my2([1,2,3],[4,5,6]);
// returns -63.0
```

CALC.SUMX2PY2(ARRAY1, ARRAY2)

Returns the sum of the *sum* of squares of corresponding values in two arrays. For arrays `[a1,a2,a3,...,an]` and `[b1,b2,b3,...,bn]`, the result gives

$$(a1^2 + b1^2) + (a2^2 + b2^2) + (a3^2 + b3^2) + \ldots + (an^2 - bn^2)$$

Parameters

array1 An array of values [a1,a2,...,an]
array2 An array of values [b1,b2,...,bn]

Example

```
CALC.sumx2py2([2,3,9,1,8,7,5],[6,5,11,7,5,4,4]);
// returns 521.0
```

CALC.SUMXMY2(ARRAY1, ARRAY2)

Returns the sum of squares of differences of corresponding values in two arrays. For arrays `[a1,a2,a3,...,an]` and `[b1,b2,b3,...,bn]`, the result gives

$$(a1-b1)^2+(a2-b2)^2+(a3-b3)^2+\ldots+(an-bn)^2$$

Parameters

array1 An array of values [a1,a2,...,an]
array2 An array of values [b1,b2,...,bn]

Example

```
CALC.sumxmy2([2,3,9,1,8,7,5],[6,5,11,7,5,4,4]);
// returns 79.o
```

CALC.TAN(NUMBER)

Returns the tangent of a number representing an angle in radians.

Parameters

number An angle in radians

Example

```
CALC.tan(pi()/4)
// returns 0.9999999999999983 (approximately 1)
```

CALC.TANH(NUMBER)

Returns the hyperbolic tangent of a real number.

Parameters

number A real number

Example

```
CALC.tanh(0.75)
// returns 0.6351489523872873
```

CALC.TRUNC(NUMBER, NUM_DIGITS)

Truncates a number to a specified number of digits.

Parameters

number Number to truncate
num_digits Number of digits (to right of decimal point) to preserve after truncation

Example

```
CALC.trunc(2245.789, 2);
// returns 2245.78
```

JS.7.5 Statistical Functions

This section presents the statistical functions provided by the CALC object.

CALC.AVEDEV(ARRAY)

Returns the average of the absolute deviations from mean of data in `array`. This is a measure of the variability in the data set.

Parameters

```
array        Array of real values for which to compute the
average absolute deviation
```

Example

```
CALC.avedev([4,5,6,7,5,4,3]);
// returns 1.0204081632653061
```

CALC.AVERAGE(ARRAY)

Returns the average (arithmetic mean) of the data in `array`.

Parameters

```
array        Array of real values to average
```

Example

```
CALC.average([4,5,6,7,5,4,3]);
// returns 4.857142857142857
```

See Also

CALC.averagea(array), to process text and Boolean values in addition to numerical values.

CALC.AVERAGEA(ARRAY)

Returns the average (arithmetic mean) of the arguments. Text and Boolean values are evaluated as follows:

– Text evaluates to 0
– `true` evaluates to 1
– `false` evaluates to 0

Parameters

```
array        Array of numerical, text, and/or Bolean values
to average
```

Example

```
CALC.averagea([1,2,3,4,'asd',true,false]);
// returns 1.5714285714285714
```

CALC.BINOMDIST(NUMBER_S, TRIALS, PROBABILITY_S, [CUMULATIVE])

Returns the individual term binomial distribution probability.

Parameters

```
number_s      Number of successes in trials
trials        Number of independent trials
probability_s Probability of success on each trial
cumulative    Boolean:
              true: Computes cumulative distribution
function (probability of at most number_s successes)
              false (default): Computes probability mass
function (probability of number_s successes)
```

Example

Probability that a fair coin tossed 10 times will land on “heads” six or fewer times.

```
CALC.binomdist(6, 10, 0.5, true);
// returns 0.828125
```

CALC.CORREL(ARRAY1, ARRAY2)

Returns the correlation coefficient of the values in `array1` and `array2`. The correlation coefficient indicates the degree of linear relationship between two variables. It ranges from -1 (perfect negative linear relationship) to +1 (perfect positive linear relationship). A value of 0 indicates no linear relationship.

Parameters

```
array1        Array of real values
array2        Array of real values
```

Example

```
CALC.correl([3,2,4,5,6],[9,7,12,15,17]);
// returns 0.9970544855015815
```

CALC.COUNT(ARRAY)

Returns the integer number of elements in `array` (both numeric and non-numeric).

Parameters

```
array         Array of numeric or non-numeric values
```

Example

```
CALC.count(['asd',true,3,2,4,5,6]);
// returns 7
```

CALC.COUNTA(ARRAY)

Counts the number of non-null elements in `array`.

Parameters

```
array         Array of numeric or non-numeric values
```

Example

```
CALC.counta(['asd',null,3,2,null,5,6]);
// returns 5
```

CALC.COUNTBLANK(ARRAY)

Counts the number of null elements in `array`.

Parameters

`array` Array of numeric or non-numeric values

Example

```
CALC.countblank(['asd',null,3,2,null,5,6]);
// returns 2
```

CALC.COUNTDISTINCT(ARRAY)

Counts the number of distinct elements in `array`.

Parameters

`array` Array of numeric or non-numeric values

Example

```
CALC.countDistinct([3,3,2,4,4,5,6]);
// returns 5
```

CALC.COUNTN(ARRAY)

Counts the number of numeric elements in `array`.

Parameters

`array` Array of numeric or non-numeric values

Example

```
CALC.countn(['asd',true,3,2,4,5,6]);
// returns 5
```

CALC.COUNTIF(ARRAY, CONDITION)

Counts the number of elements in the array that meet a certain condition, such as "`>100`" or "`==21`" or "`=='desk'`".

Parameters

`array` Array of numeric or non-numeric values
`condition` Condition against which elements should be compared

Example

```
CALC.countif([3,2,4,5,6], '> 4');
// returns 2

CALC.countif(['me','you','you','me','me'], '=="me"'))
// returns 3
```

CALC.COVAR(ARRAY1, ARRAY2)

Returns the covariance between the values in `array1` and `array2`. Covariance is the average of the products of deviations for each data point pair.

Parameters

array1	Array of real values
array2	Array of real values

Example

```
CALC.covar([3,2,4,5,6],[9,7,12,15,17]);
// returns 5.2
```

CALC.DEVSQ(ARRAY)

Returns the sum of squares of deviations of values in `array` from their sample mean.

Parameters

array	Array of real values

Example

```
CALC.devsq([3,2,4,5,6]);
// returns 10.0
```

CALC.EXPONDIST(X, LAMBDA, CUMULATIVE)

Returns the exponential distribution. Use this function to model the time between events. `X` is the value of the function. `Lambda` is the parameter value. `Cumulative` is a logical value that indicates which form of the exponential function to provide. If `Cumulative` is true, the function returns the cumulative distribution function; if false, it returns the probability density function

Example

```
CALC.expondist(0.2, 10, true);
// returns 0.864665
```

CALC.FISHER(X)

Returns the Fisher transformation at numeric value `x`. This transformation produces a function that is normally distributed rather than skewed. Use this function to perform hypothesis testing on the correlation coefficient.

Example

```
CALC.fisher(0.75);
// returns 0.97295
```

CALC.FISHERINV(X)

Returns the inverse of the Fisher transformation. `X` is a numeric value for which you want the inverse transformation.

CALC.FORECAST(X, KNOWN_Y_VALUES, KNOWN_X_VALUES)

Calculates, or predicts, a future value by using existing values. The predicted value is a y-value for a given x-value. The known values are existing x-values and y-values and the new value is predicted by using linear regression. `X` is the data point for which you want to predict a value. `Known_y_values` is the dependent array or range of data. `Known_x_values` is the independent array or range of data.

Example

```
CALC.forecast(30, [6,7,9,15,21],[20,28,31,38,40]);
// returns 10.6072
```

CALC.FREQUENCY(DATA_ARRAY, BINS_ARRAY)

Calculates how often values occur within a range of values and then returns an array of numbers. `Data_array` is an array of or reference to a set of values for which you want to count frequencies. `Bins_array` is an array of or reference to intervals into which you want to group the values in `Data_array`.

Example

```
CALC.frequency([79,85,78,85,50,81,95,88,97],[70,79,89]);

// returns an array of length 4. Say the name of the array
is ‘freq’

// freq[0] = 1 (number of elements less than or equal to70)
// freq[1] = 2 (number of elements in range 70 -79)
// freq[2] = 4 (number of elements in range 79 -89)
// freq[3] = 2 (number of elements greater than or equal to
90)
```

CALC.GEOMEAN(ARRAY)

The geometric mean of an array.

CALC.HARMEAN(ARRAY)

Returns the harmonic mean of a data set. The harmonic mean is the reciprocal of the arithmetic mean of reciprocals.

CALC.HYPGEOMDIST(SAMPLE_S, NUMBER_SAMPLE, POPULATION_S, NUMBER_POPULATION)

Returns the hypergeometric distribution, the probability of a given number of sample successes, given the sample size, population successes and

population size. `Sample_s` is the number of successes in the sample. `Number_sample` is the size of the sample. `Population_s` is the number of successes in the population. `Number_population` is the population size.

Example

```
CALC.hypgeomdist(1, 4, 8, 20);
// returns 0.36326
```

CALC.INTERCEPT(ARRAYY, ARRAYX)

Calculates the point at which a line will intersect the y-axis by using existing x-values and y-values. The intercept point is based on a best-fit regression line plotted through the known x-values and known y-values.

Example

```
CALC.intercept([8,11,14,17],[1,2,3,4]);
// returns 5
```

CALC.KURT(ARRAY)

Returns the kurtosis of a data set. Kurtosis characterizes the relative peakedness or flatness of a distribution compared with the normal distribution.

CALC.LARGE(ARRAY, K)

Returns the *k*th largest value in a data set.

CALC.MAX(ARRAY)

Returns the largest value in the array.

CALC.MAXA(ARRAY)

Returns the largest value in the array. Booleans are considered to be 1 if true and zero if false. Text representations of numbers are considered.

CALC.MEDIAN(ARRAY)

Returns the median of the elements in the array. Median is the value in the middle.

Example

```
CALC.median([1,2,3,4,5,6,7]);
// returns 4
```

CALC.MIN(ARRAY)

Returns the smallest value in the array.

CALC.MINA(ARRAY)

Returns the smallest value in the array. Booleans are considered to be 1 if true and zero if false. Text representations of numbers are considered.

CALC.MODE(ARRAY)

Returns the most frequently occurring, or repetitive, value in an array or range of data.

Example

```
CALC.mode([1,1,2,3,1,4,5,6,7]);
// returns 1
```

CALC.NEGBINOMDIST(NUMBER_F, NUMBER_S, PROBABILITY_S)

Returns the negative binomial distribution, the probability that there will be `number_f` failures before the `number_s`-th success, when the constant probability of a success is `probability_s`.

Example

```
CALC.negbinomdist(10, 5, 0.25);
// returns 0.05504
```

CALC.PEARSON(ARRAY1, ARRAY2)

Returns the Pearson product moment correlation coefficient, r, a dimensionless index that ranges from -1.0 to 1.0 inclusive and reflects the extent of a linear relationship between two data sets.

Example

```
CALC.pearson([9,7,5,3,1],[10,6,1,5,3]);
// returns 0.6993
```

CALC.PERCENTILE(ARRAY, K)

Returns the k-th percentile of values in a range. You can use this function to establish a threshold of acceptance.

Example

```
CALC.percentile([9,7,5,3,1], 0.9);
// returns 8.2 (90 percentile)
```

CALC.PERCENTRANK(ARRAY, X, SIGNIFICANCE)

Array is the array or range of data with numeric values that defines relative standing. X is the value for which you want to know the rank. Significance is an optional value that identifies the number of significant digits for the returned percentage value.

Example

```
CALC.percentrank([13,12,11,8,4,3,2,1,1,1],2,3);
// returns 0.333
```

(percent rank of the element 2 in the list above)

CALC.PERMUT(NUMBER, NUMBER_CHOSEN)

Returns the number of permutations for a given number of objects that can be selected from number objects. A permutation is any set or subset of objects or events where internal order is significant. `Number` is an integer that describes the number of objects. `Number_chosen` is an integer that describes the number of objects in each permutation.

CALC.POISSON(X, MEAN, CUMULATIVE)

Returns the Poisson distribution. A common application of the Poisson distribution is predicting the number of events over a specific time. `X` is the number of events. `Mean` is the expected numeric value. `Cumulative` is a Boolean value that determines the form of the probability distribution returned. If `Cumulative` is true, the function returns the cumulative Poisson probability that the number of random events occurring will be between zero and `x` inclusive; if false, it returns the Poisson probability mass function that the number of events occurring will be exactly `x`.

Example

```
CALC.poisson(2, 5, true);
// returns 0.124652
```

CALC.PROB(ARRAYX, ARRAYPROB, LOWER_LIMIT, UPPER_LIMIT)

Returns the probability that values in a range are between two limits. `Arrayx` is the range of numeric values of x with which there are associated probabilities. `Arrayprob` is a set of probabilities associated with values in `Arrayx`. `Lower_limit` is the lower bound on the value for which you want a probability. `Upper_limit` is the optional upper bound on the value for which you want a probability if not supplied the function returns the probability that values in `Arrayx` are equal to `Lower_limit`.

Example

```
CALC.prob([0,1,2,3], [0.2,0.3,0.1,0.4], 1, 3);
// returns 0.8 (probability that x lies within 1 and 3)
```

CALC.QUARTILE(ARRAY, QUART)

Returns the quartile of a data set. Argument `array` is the array of numeric values for which you want the quartile value. Argument `quart` (0 to 4) indicates which value to return.

Example

```
CALC.quartile([1,2,4,7,8,9,10,12], 1);
// returns 3.5 (first quartile (25 percentile))
```

CALC.RANK(NUMBER, ARRAY, ORDER)

Returns the rank of a number in a list of numbers. The rank of a number is its size relative to other values in a list. `Number` is the number whose rank you want to find. `Array` is an array numbers. Nonnumeric values in `Array` are ignored. `Order` is a number (0 or 1) specifying how to rank number. (0 = from top, 1 = from bottom).

Example

```
CALC.rank(7,[1,2,4,7,8,9,10,12],1);
// returns 4; the value '7' is fourth from bottom

CALC.rank(7,[1,2,4,7,8,9,10,12],0);
// returns 5; the value '7' is fifth from top
```

CALC.RSQ(KNOWN_Y_VALUES, KNOWN_X_VALUES)

Returns the square of the Pearson product moment correlation coefficient through data points in `known_y_values` and `known_x_values`.

CALC.SKEW(ARRAY)

Returns the skewness of a distribution. Skewness characterizes the degree of asymmetry of a distribution around its mean.

Example

```
CALC.skew([3,4,5,2,3,4,5,6,4,7]);
// returns 0.35954
```

CALC.SLOPE(ARRAYY, ARRAYX)

Returns the slope of the linear regression line through data points in `arrayx` and `arrayy`. `arrayx` is an array or cell range of numeric independent data points. `arrayy` is an array or cell range of numeric dependent data points.

CALC.SMALL(ARRAY, K)

Returns the k-th smallest value in a data set. `Array` is an array or range of numerical data for which you want to determine the k-th smallest value. `K` is the position (from the smallest) in the array of data to return.

CALC.STANDARDIZE(X, MEAN, STANDARD_DEV)

Returns a normalized value from a distribution characterized by mean and standard_dev. `X` is the value you want to normalize. `Mean` is the arithmetic mean of the distribution. `Standard_dev` is the standard deviation of the distribution.

Example

```
CALC.standardize(42, 40, 1.5);
// returns 1.3333 (standardized value for 42)
```

CALC.STDEV(ARRAY)

Estimates standard deviation based on a range of data. The standard deviation is a measure of how widely values are dispersed from the average value. The standard deviation is calculated using the “unbiased” or “n-1” method. Text and Boolean values are ignored.

Example

```
CALC.stdev(
[1345,1301,1368,1322,1310,1370,1318,1350,1303,1299]);
// returns 27.46391
```

CALC.STDEVA(ARRAY)

Estimates standard deviation based on a range of data. The standard deviation is calculated using the “unbiased” or “n-1” method. Text values are ignored. Boolean values are considered to be 1 (true) or 0 (false).

Example

```
CALC.stdeva(
[true, false,1345,1301,1368,1322,1310,
                              1370,1318,1350,1303,1299]);
// returns 517.5588
```

CALC.STDEVP(ARRAY)

Calculates standard deviation based on the entire population. The standard deviation is calculated using the “biased” or “n” method.

Example

```
CALC.stdevp(
[1345,1301,1368,1322,1310,1370,1318,1350,1303,1299]);
// returns 26.05455
```

CALC.STDEVPA(ARRAY)

Calculates standard deviation based on the entire population. The standard deviation is calculated using the “biased” or “n” method. Text values are ignored. Boolean values are considered to be 1 (true) or 0 (false).

CALC.STEYX(ARRAYY, ARRAYX)

Returns the standard error of the predicted y-value for each x in the regression. `Arrayy` is an array or range of dependent data points. `Arrayx` is an array or range of independent data points.

Example

```
CALC.steyx([2,3,9,1,8,7,5], [6,5,11,7,5,4,4]);
```

```
// returns 3.3057
```

CALC.TRIMMEAN(ARRAY, PERCENT)

Returns the mean of the interior of a data set by excluding a percentage of data points from the top and bottom tails of a data set. You can use this function when you wish to exclude outlying data from your analysis. `Array` is the array or range of values to trim and average. `Percent` is the fractional number of data points to exclude from the calculation.

Example

```
CALC.trimmean([4,5,6,7,2,3,4,5,1,2,3], 0.2);
// returns 3.777
(mean of the data set with 20% of the values eliminated, 10%
from front and 10% from the back)
```

CALC.VARN(ARRAY)

Estimates variance based on a sample. Sum of the squares of the deviations from the mean of each sample divided by number of samples -1. Text and Boolean values are ignored.

CALC.VARA(ARRAY)

Estimates variance based on a sample. Text values are ignored. Boolean values are considered to be 1 (true) or 0 (false).

Example

```
CALC.vara(
[1345,1301,1368,1322,1310,1370,1318,1350,1303,1299]);
// returns 754.266
```

CALC.VARP(ARRAY)

Estimates variance based on the entire population. Sum of the squares of the deviations from the mean of each sample divided by number of samples.

Example

```
CALC.varp(
[1345,1301,1368,1322,1310,1370,1318,1350,1303,1299]);
// returns 678.8399
```

JS.7.6 Text Functions

This section presents the text functions provided by the CALC object.

CALC.CHARACTER(NUMBER)

Returns the character specified by a number.

Example

```
CALC.character(77);
// returns M
```

CALC.CODE(STRING)

Returns a numeric code for the first character in a text string.

Example

```
CALC.code('jsd');
// returns 106
```

CALC.CONCATENATE(ARRAY)

Joins several text strings in the array into one single text string.

Example

```
CALC.concatenate(['madam','i','am','adam']);
// returns madamiamadam
```

CALC.DOLLAR(NUMBER, DECIMALS)

Converts a number to text format and applies a currency symbol. Decimals is the number of digits to the right of the decimal point.

CALC.EXACT(STRING1, STRING2)

Compares two text strings and returns TRUE if they are exactly the same, FALSE otherwise. EXACT is case-sensitive but ignores formatting differences.

CALC.FIND(STRING1, STRING2, SEARCH_INDEX)

Finds the location of the first occurrence of `string1` within `string2`, starting the search from position specified by `search_index`. (The first character is at index 1).

Example

```
CALC.find('asd','asdqwertasd', 6);
// returns 9
```

CALC.FIXED(NUMBER, DECIMALS, NO_COMMAS)

Rounds a number to the specified number of decimals, formats the number in decimal format using a period and commas and returns the result as text. `Number` is the number you want to round and convert to text. `Decimals` is the number of digits to the right of the decimal point. `No_commas` is a Boolean value that, if true, prevents the function from including commas in the returned text.

Example

```
CALC.fixed(1234.567, 2, false);
// returns 1,234.57
```

CALC.LEFT(STRING, NUM_CHARS)

Returns `num_char` number of characters from the left of a text string.

Example

```
CALC.left('abcdefghij', 4);
// returns abcd
```

CALC.LEN(STRING)

Returns the number of characters in a text string.

CALC.LOWER(STRING)

Converts all uppercase characters in a string to lowercase.

CALC.MID(STRING, START_NUM, NUM_CHARS)

Returns a specific number of characters from a text string, starting at the position you specify, based on the number of characters you specify.

Example

```
CALC.mid('abcdefghij', 2, 4);
// returns bcde
```

CALC.PROPER(STRING)

Capitalizes the first letter of all the words in a text string.

Example

```
CALC.proper('my name is alex');
// returns 'My Name Is Alex'
```

CALC.REPLACE(OLD_STRING, START_NUM, NUM_CHARS, NEW_STRING)

Replaces part of a text string, based on the number of characters you specify, with a different text string. `Old_string` is text in which you want to replace some characters. `Start_num` is the position of the character in `Old_string` that you want to replace with `New_string`. `Num_chars` is the number of characters in `Old_string` that you want to replace with `New_string`.

Example

```
CALC.replace('abcdefghijkl',6,5,'*');
// returns abcde*kl
(replaces 5 characters of the string with '*' starting from
the 6th character)
```

CALC.REPT(TEXT, NUMBER_TIMES)

Repeats text a given number of times (concatenates the text to itself).

CALC.RIGHT(STRING, NUM_CHARS)

Returns `num_char` number of characters from the right of a text string. See left() for sample script.

CALC.SEARCH(FIND_STRING, WITHIN_STRING, START_NUM)

Locates one text string within a second text string and returns the starting position of the first text string from the first character of the second text string.(The first character of a string has index 1.)

Example

```
CALC.search('e','statements', 6);
// returns 7
```

CALC.SUBSTITUTE(TEXT, OLD_TEXT, NEW_TEXT, INSTANCE_NUM)

Substitutes `New_text` for `Old_text` in a text string. Use this function when you want to replace specific text in a text string; use replace() when you want to replace any text that occurs in a specific location in a text string. `Text` is the text or the reference to a cell containing text for which you want to substitute characters. `Instance_num` specifies which occurrence of `Old_text` you want to replace.

Example

```
CALC.substitute('asd/pqr/xyz','/','-', 1);
// returns asd-pqr/xyz (substitutes instance 1 of '/' with
'-')
```

CALC.T(VALUE)

Returns the text referred to by `Value`. `Value` can be any type of data if it is text data then the functions returns the parameter passed, if it is numerical or Boolean data then the function returns an empty string.

Example

```
CALC.t('asd');
// returns asd

CALC.t(true); or var t = CALC.t(56);
// returns ''
```

CALC.TEXT(VALUE, FORMAT_TEXT)

Converts a numeric value to text based on a certain format.

Example

```
CALC.text(2800, '$0.00');
// returns '$2800.00'
```

CALC.TRIM(STRING)

Removes all white spaces from a string except single spaces between words.

CALC.UPPER(STRING)

Converts string to uppercase.

CALC.VALUE(STRING)

Parses a string to a numeric value.

JS.8 Regular Expression Object Functions

A regular expression object contains the pattern of a regular expression. It has properties and methods for using the regular expression to find and replace matches in strings. A regular expression can be created as a literal:

```
re = /.*JavA.*/i;
```

Alternatively, it can be used to created using the RegExp constructor:

```
re = new RegExp('.*JavA.*', 'i');
```

A regular expression can be used in the String.match() or String.search() method to find the regular expression in a string, or evaluated against a string using one of the RegExp methods.

COMPILE(PATTERN[, FLAGS])

Compile a regular expression.

EXEC(STR)

Execute the search for a match in a string and returns a result array

TEST(STR)

Execute the search for a match in a string and returns true if the match is found and false otherwise.

Examples

```
var re = new RegExp(".*JavA.*", 'i');
var found = re.test(str);
```

JS.9 Java Objects (LiveConnect)

The JavaScript engine used in Style Intelligence supports the *LiveConnect* feature, which allows a JavaScript script to access Java classes and objects. This is particularly useful in Style Intelligence because the host environment is running inside a Java virtual machine and many properties of the report elements require Java objects.

A Java object can be created using the 'new' operator.

```
Text1.foreground = new java.awt.Color(0xFF0000);
```

The Java class name must be fully qualified in the 'new' expression. Once a Java object is created, its properties and methods can be accessed from the script. The properties are discovered by search for the getter and setter methods etc. A getter method is a non-void method starting with 'get' and has an empty parameter list. A setter method is a void method starting with 'set' and accepts a single parameter. The name following the 'get' and 'set' is used as the property name with the first letter converted to lowercase.

If property has only a getter method, it is marked as a read-only property. Assignment to the property is ignored.

```
var rgb = Text1.foreground.red; // Color.getRed()
```

Other public methods in the Java object are accessible as JavaScript methods in a script.

```
Text1.foreground = Text1.foreground.darker();
// Color.darker()
```

JS.10 JavaScript Statements

JavaScript statement syntax is very similar to Java statement syntax. Some important language constructs are introduced in the following sections:

- *Variable Definition*
- *Function Definition*
- *The 'if' Statement*
- *The 'try-catch' Statement*
- *Iteration Statements*
- *The 'switch' Statement*
- *The 'with' Statement*

JS.10.1Variable Definition

A variable statement inside a function declares a local variable. A variable statement outside of a function body declares a global variable.

```
var val;
var array1 = [1, 2, 3];
```

JS.10.2Function Definition

The ‘function’ keyword introduces a new function in the global scope. Once a function is defined, it can be called in other parts of the script.

```
function sum(array1) {
  var total = 0;

  for(var idx = 0; idx < array1.length; idx++) {
    total += array1[idx];
  }

  return total;
}

var total = sum(array2);
```

JS.10.3The ‘if’ Statement

The *if* conditional structure executes the statements within the ‘if’ block only when the specified condition is true. Alternative conditions can be tested by using one or more optional ‘else if’ blocks. An optional final ‘else’ block will be executed if all other conditions evaluate false.

```
if(condition0) {
  statements to execute if condition0 is true
}
else if(condition1) {
  statements to execute if condition1 is true
}
else if(condition2) {
  statements to execute if condition2 is true
}
else if(condition3) {
  statements to execute if condition3 is true
}
else {
  statements to execute if all above conditions are false
}
```

See Also

Appendix JS.10.6, *The ‘switch’ Statement*, to conveniently test a larger set of conditions.

JS.10.4The ‘try-catch’ Statement

The “try-catch” construction allows you to catch a script error before it causes report execution to fail. To catch an error, place the desired sequence of code inside a “try” block. If any line of this “try” code generates an error, program execution immediately jumps to the code in the “catch” block.

```
try {
  // Code to try
}
catch(err) {
```

```
  // Alternative code for case of error
}
finally {
  // Code to execute in either case
}
```

The “err” parameter contains the exception message that was generated by the try block. The “finally” block contains code to be executed whether an error occurred or not. The “catch” block and “finally” block are optional, but cannot both be omitted.

JS.10.5 Iteration Statements

A ‘for’ loop consists of three optional expressions, enclosed in parentheses and separated by semi-colons.

```
for([init-expression];[condition];[increment-expression]) {
  statements;
}
```

The initial-expression is evaluated before the loop starts. It is normally used to declare and initialize a counter variable. The condition statement is evaluated at the beginning of every loop. If the condition is true, the statements are executed. Otherwise, the loop terminates. The increment-expression is executed after the body statements. It is normally used to increment the counter variable.

```
for(var idx = 0; idx < array1.length; idx++) {
  ...
}
```

A special ‘for’ loop allows iteration through all properties of an object. If the object is an array, each item in the array is treated as a property.

```
var props = "";
for(var item in obj) {
  props += item + " = " + obj[item];
}
```

A ‘while’ statement is a loop with a single condition. The condition is evaluated at the beginning of every loop. If the condition is true, the body of the loop is executed. Otherwise, the loop is terminated.

```
while(condition) {
  statements;
}
```

A slight variation of the ‘while’ statement is ‘do...while’. Instead of evaluating the condition at the beginning of the loop, ‘do...while’ statement evaluates the condition at the end of a loop. This guarantees the loop is at least executed once.

```
do {
  statements
```

```
} while(condition);
```

A loop can be terminated in the body by inserting a ‘break’ statement. A ‘break’ statement causes the execution to immediately transfer to the point following the end of the loop.

Similarly, a ‘continue’ statement interrupts the execution of a loop and transfer the execution to the end of the loop block.

JS.10.6 The ‘switch’ Statement

A ‘switch’ statement evaluates an expression and matches its result with a list of choices. If a match is found, the statements under the choice are executed. Otherwise, if a ‘default’ label exists, the statements in the default block are executed.

```
switch(expression) {
case label:
  statements;
  break;
case label:
  statements;
  break;
  ...
default:
  statements;
}
```

See Also

Appendix JS.10.3, *The ‘if’ Statement*, to conveniently test a small set of conditions.

JS.10.7 The ‘with’ Statement

A ‘with’ statement establishes an object as the default object for a set of statements.

```
with(object) {
  statements;
}
```

The properties and methods of the object can be referenced without qualifying the names with the object name.

```
with(Text1) {
  text = "contents of this text element";
}
```

JS.11 JavaScript Name/Value Pairs

You can use the following forms to create objects in JavaScript:

- color: `{r:255, g:255, b:255}`
- insets: `{top:0, left:0, bottom:0, right:0}`

- dimension: `{width:2, height:2}`
- point: `{x:1, y:1} or {row:3, column:2}`
- position: `{x:3.3, y:1.2}`
- size: `{width:2.5, height:3.0}`
- hyperlink:

```
{link:"report2", target:"reportFrame", tooltip:"Hello",
type:"report", sendReportParameters:true, state:"NJ",
date:new Date()}
```

The `type` can be a report, web, or archive, and parameters can be added as name/value pairs, e.g., state, date.

- shape: `{type:rectangle, x:0, y:0, width:20, height:30}`; Here, type can be an ellipse.

JS.12 JavaScript Operators

The operators in JavaScript are almost identical to the operators in Java:

JS.12.1Assignment Operators

An assignment operator assigns a value to its left operand based on the value of its right operand.

```
x = y;
```

Arithmetic operators and bitwise operators can be combined with the assignment operator as a short hand for performing a computation and assigning the results to a variable.

```
x += y;
```

JS.12.2Comparison Operators

A comparison operator compares two values and returns ‘true’ or ‘false’ depending on the comparison result.

OPERATOR	DESCRIPTION
==	Equal
!=	Not equal
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to

JS.13 StyleConstant Object

The StyleConstant object is a static object that contains all definitions for constants used in the Style Intelligence host environment. These constants

can be accessed without creating an instance of a StyleConstant object, as shown below:

```
line = StyleConstant.DOUBLE_LINE
```

ALIGNMENTS

Alignment constants are defined for horizontal or vertical directions. By using a bitwise OR of the values, the alignment options can be combined.

ALIGNMENT	DESCRIPTION
H_LEFT	Left alignment
H_CENTER	Horizontal centered alignment
H_RIGHT	Right alignment
V_TOP	Top alignment
V_CENTER	Vertical centered alignment
V_BOTTOM	Bottom alignment
V_BASELINE	Character baseline alignment
FILL	Fill in whole space
LEFT	Synonym for H_LEFT
CENTER	Synonym for H_CENTER
RIGHT	Synonym for H_RIGHT

Type

```
integer
```

Example

```
alignment = StyleConstant.H_RIGHT | StyleConstant.V_CENTER;
```

See Also

Alignment Property, in *Report Scripting*, for more information on setting report alignments.

Alignment Property, in *Advanced Dashboard Scripting*, for more information on setting Viewsheet alignments.

CHART STYLES

The chart style constants specify how a chart presents its data. It is recommended to access these constants through the *Chart Styles* property of the *Chart Constants*.

CHART STYLE	DESCRIPTION
CHART_AUTO	Automatic style selection
CHART_PARETO	Pareto chart.
CHART_LINE	Line chart
CHART_LINE_STACK	Stacked line chart
CHART_POINT	Point chart
CHART_BAR	2D bar chart

CHART_BAR_STACK	Stacked bar chart
CHART_PIE	2D pie chart
CHART_3D_BAR	3D bar chart
CHART_3D_BAR_STACK	3D stacked bar chart
CHART_3D_PIE	3D pie chart
CHART_STOCK	High-low-closing chart
CHART_AREA	Area chart
CHART_AREA_STACK	Stacked area chart
CHART_RADAR	Radar chart
CHART_FILL_RADAR	Filled radar chart
CHART_CANDLE	Candle chart
CHART_WATERFALL	Waterfall chart

Type

`integer`

LINE STYLES

The line style constants can be used in all properties requiring line style constants, such as separator style, tab fill style, and table borders.

LINE STYLE	DESCRIPTION
NO_BORDER	No line
ULTRA_THIN_LINE	Very thin line at ¼ of a point width
THIN_THIN_LINE	Very thin line at ½ of a point width
THIN_LINE	Thin line at one point width
MEDIUM_LINE	Medium width line
THICK_LINE	Thick width line
DOUBLE_LINE	Double line
RAISED_3D	Raised 3D line
LOWERED_3D	Lowered 3D line
DOUBLE_3D_RAISED	Raised double 3D line
DOUBLE_3D_LOWERED	Lowered double 3D line
DOT_LINE	Dotted line
DASH_LINE	Dash line
MEDIUM_DASH	Medium size dash line
LARGE_DASH	Large size dash line
BREAK_BORDER	A special line style that signals a page break in a table row. This line style can only be used in the table border.

Type

`integer`

Example

```
style = StyleConstant.DOT_LINE;
```

PAINTER LAYOUT

Painter element layout controls how a painter element is laid out when it is bigger than the remaining space of a page.

LAYOUT	DESCRIPTION
PAINTER_NON_BREAK	If a painter is larger than the remaining page, advance to the next page. If the painter is larger than the entire page, draw the top portion in a page and ignore the remaining part.
PAINTER_BREAKABLE	If a painter is larger than the remaining page, print the portion that can fit in the available space and continue to the next page for the remaining part.

Type

```
integer
```

Example

```
layout = StyleConstant.PAINTER_BREAKABLE;
```

POINT SHAPES

The point shape constants can be used to specify the shape used in a point chart.

```
CIRCLE
TRIANGLE
SQUARE
CROSS
STAR
DIAMOND
X
FILLED_CIRCLE
FILLED_TRIANGLE
FILLED_SQUARE
FILLED_DIAMOND
```

Type

```
integer
```

Example

```
chartDescriptor.plotDescriptor.setPointStyle(0,
StyleConstant.FILLED_CIRCLE);
```

TABLE LAYOUT

Table layout affects the table size calculation and table placement on a report.

LAYOUT	DESCRIPTION
TABLE_FIT_CONTENT	Calculate the column width to fit the largest cell in each column
TABLE_FIT_PAGE	Resize a table to fit the width of a page
TABLE_EQUAL_WIDTH	Assign the same width to each column
TABLE_FIT_CONTENT_1PP	Same as TABLE_FIT_CONTENT, but only put one table region on each page
TABLE_FIT_CONTENT_PAGE	Same as TABLE_FIT_CONTENT, but resize the table segments to fit the width of a page after table has been broken up into regions.

Type

```
integer
```

Example

```
layout = StyleConstant.TABLE_FIT_CONTENT_PAGE;
```

TEXTBOX SHAPES

The TextBox shape constants can be used to change the shape of a textbox.

```
BOX_RECTANGLE
BOX_ROUNDED_RECTANGLE
```

Type

```
integer
```

Example

```
shape = StyleConstant.BOX_ROUNDED_RECTANGLE;
```

WRAPPING STYLE

Wrapping style defines how contents flow around a painter object.

WRAPPING STYLE	DESCRIPTION
WRAP_NONE	No wrapping. The painter element is printed as a background. The content flows on top of this painter.
WRAP_LEFT	Allow contents to wrap around the painter on the left.
WRAP_RIGHT	Allow contents to wrap around the painter on the right.

WRAP_BOTH	Allow contents to wrap on both sides of the painter.
WRAP_TOP_BOTTOM	No wrapping around either sides of the painter.

Type

`integer`

Example

```
wrapping = StyleConstant.WRAP_RIGHT;
```

APPENDIX CT:Chart Script Tutorial

This appendix provides a set of chart examples to illustrate some common chart scripting tasks, such as color-coding data, changing labels and axes, and positioning chart annotations.

The scripts in this appendix can be used for both report Charts (using Style Studio) and Viewsheet Charts (using Visual Composer). However, there are some differences in how charts are configured in these different environments. See the references for further information.

See Also
Advanced Dashboard Scripting, for information on scripting for Viewsheets.
Report Scripting, for information on scripting for Reports.
Creating a Chart Using API Functions, for basic chart setup.
Appendix CR:*Chart Script Reference*, for complete information on all API functions.

CT.1 Modifying Chart Properties

To modify the basic properties of a Chart element, simply specify the desired property value in the *onLoad* handler, *onRefresh* handler, or component script. The following sections provide some basic examples of property modifications.

Whenever possible, use the syntax auto-completion feature to enter legitimate property names and values. (See *Editing Script* for reports, or *Adding Script to a Viewsheet* for dashboards.)

See Also
onLoad Handler, in *Report Scripting*, for information on the report handler.
onRefresh Handler, in *Advanced Dashboard Scripting*, for information on the Viewsheet handler.
Element Script, in *Report Scripting*, for information on scripting report elements
Adding Component Script, in *Advanced Dashboard Scripting*, for information on scripting Viewsheet elements

CT.1.1Modifying the Chart Style

To modify the style of a chart in script (e.g., line, bar, pie, etc.), use the *chartStyle* attribute:

You can switch between multiple-style and single-style in the 'Select Chart Style' panel.

- For a multi-style chart, use `chartStyle` together with the name of the dataset you want to style, e.g., `Chart1.chartStyle['Sum(Total)']`.
- For a single-style chart, use `chartStyle` without any modifiers, e.g., `Chart1.chartStyle`.

For example, if you want to parameterize the style of a chart, you could prompt the user for a parameter called ‘Chart Style’. Then access this parameter in the chart script and modify the chart style accordingly.

```
if (parameter['Chart Style'] == 'bar') {
   Chart1.chartStyle=Chart.CHART_BAR;
}
else if (parameter['Chart Style'] == 'line') {
   Chart1.chartStyle=Chart.CHART_LINE;
}
```

CT.1.2 Modifying Axis Title Text

To modify axis title text, use *xTitle.text* (`x2Title.text`) and *yTitle.text*, (`y2Title.text`).

Use auto-complete for correct syntax.

```
Chart1.xTitle.text = 'Text to go below bottom X-axis'
Chart1.x2Title.text = 'Text to go above top X-axis'
```

CT.1.3 Modifying Axis Properties

To modify axis properties, including labels, visibility, tick marks, etc., use the “`axis`” properties (*axis.font*, *axis.minimum*, *axis.labelColor*, *axis.format*, etc.).

Use auto-complete for correct syntax. Type “.” after “axis” to see prompt.

```
Chart1.axis.Employee.font = 'Comic Sans MS-BOLD-12'
Chart1.axis.Employee.ticksVisible = false
Chart1.axis.Employee.labelColor = [0,0,255]
Chart1.axis['Sum(Total)'].logarithmic = true
Chart1.axis['Sum(Total)'].minimum = 10000
Chart1.axis['Sum(Total)'].format = [Chart.DECIMAL_FORMAT,
"#,###.00"];
```

See Also

Appendix CR.1.2 , *General Axis Properties*, for general axis characteristics.
Appendix CR.1.3 , *Measure Axis Properties*, for numeric axis characteristics.
Appendix CR.1.4 , *Title Properties*, for axis title characteristics.
Appendix CR.1.5 , *Plot Properties*, for plot area characteristics.

CT.2 Modifying a Chart Data Binding

The `bindingInfo` property of a Chart provides access to the data binding options in the Chart’s ‘Data Binding’ dialog box. This includes the binding of fields to the X-axis, Y-axis, and VisualFrames (Color, Shape, etc.), as well as grouping and summarization settings.

Walkthrough

In the following example, you will bind a chart to the ‘All Sales’ query, set the axis bindings, define a subseries, and set grouping and summarization. Follow the steps below:

1. Create a new report or Viewsheet. (For a Viewsheet, select the ‘All Sales’ query as the data source.)

2. Add a new Chart element to the report or Viewsheet.

Note: bindingInfo modifications must be made prior to query execution. Therefore, attach bindingInfo scripts to the *onLoad* or *onRefresh* handler.

3. Open the Script Editor for the *onLoad Handler*:

   a. For a report, right-click an empty area of the report and select 'Script' from the context menu.

   b. For a Viewsheet, press the 'Options' button in the toolbar to open the 'Viewsheet Options' dialog box, then click the **Script** tab.

Note: The Viewsheet must be based on the 'All Sales' query.

4. In the Script Editor, enter the commands below to bind the chart to the 'All Sales' query.

```
Chart1.query = "All Sales";
```

5. Specify query fields to bind to the X-axis ('Company') and Y-axis ('Total'):

```
Chart1.bindingInfo.xFields = [["Company",Chart.STRING]];
Chart1.bindingInfo.yFields = [["Total",Chart.NUMBER]];
```

6. Set the aggregation/summarization method:

```
Chart1.bindingInfo.setFormula("Total",Chart.MAX_FORMULA)
```

7. Set the desired grouping options (e.g., top-N filtering):

```
Chart1.bindingInfo.setTopN("Company",5)
Chart1.bindingInfo.setTopNReverse("Company",false)
Chart1.bindingInfo.setTopNSummaryCol("Company",
   "Max(Total)");
```

8. Specify the query fields to bind to a "VisualFrame," (i.e., for visual coding as subseries):

```
Chart1.bindingInfo.setColorField("Employee",Chart.STRING);
```

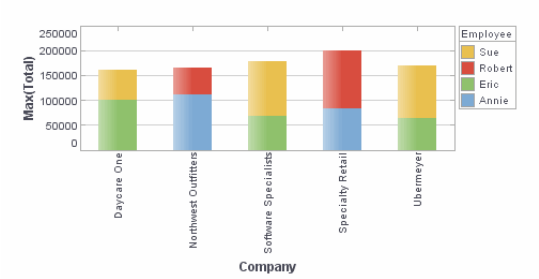
9. Close the Editor, an preview the report.

The complete script is shown below:

```
Chart1.query = "All Sales";

Chart1.bindingInfo.xFields = [["Company",Chart.STRING]];
Chart1.bindingInfo.yFields = [["Total",Chart.NUMBER]];
Chart1.bindingInfo.setFormula("Total",Chart.MAX_FORMULA)
Chart1.bindingInfo.setTopN("Company",5)
Chart1.bindingInfo.setTopNReverse("Company",false)
```

```
Chart1.bindingInfo.setTopNSummaryCol("Company","Max(Total)"
);
Chart1.bindingInfo.setColorField("Employee",Chart.STRING);
```



See Also

bindingInfo.xFields, *bindingInfo.yFields*
bindingInfo.setFormula(col, formula, binding)
bindingInfo.setTopN(col,N), *bindingInfo.setTopNReverse(col,boolean)*
bindingInfo.setTopNSummaryCol(col,agg)
bindingInfo.setColorField(col, type)
Bind Chart to Query or Worksheet (Report/Viewsheet), to reference an arbitrary query or Worksheet.
Binding Data to a Chart Element, in *Advanced Report Design*, for information on the 'Data Binding' dialog box.
Modifying a Chart Element using API Functions, to modify other aspects of an existing chart.
Creating a Chart Using API Functions, to script a chart from the ground-up.
Appendix CR.1.6 , *The Chart bindingInfo Property*, for all binding properties.

CT.3 Modifying a Chart Element using API Functions

The Chart API provides access to charting engine commands using Java object syntax. You can use these commands to directly modify the graphical elements displayed by a Chart.

Walkthrough

In this example, you will first bind a chart to a data source, and then make further modifications to the chart display by using Chart API commands. Follow the steps below:

1. Create a new report or Viewsheet. (For a Viewsheet, select the 'All Sales' query as the data source.)

2. Add a new Chart element to the report or Viewsheet.

3. For a report, click on the Chart to open the Chart Editor. For a Viewsheet, press the 'Edit' button at the top-right of the Chart.

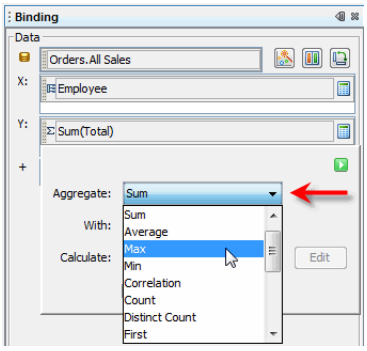
4. For a report, in the **Data** panel, expand the 'Orders' data source and the 'All Sales' query. For a Viewsheet, in the **Data Source** panel, expand the 'All Sales' data block.

5. Drag the 'Employee' field to the 'X' region.

6. Follow the steps below to bind the 'Total' field so that it provides three different measures (maximum, minimum, and average) on the chart:

   a. From the 'All Sales' source, drag the 'Total' field to the 'Y' region.

   b. Click the 'Edit Measure' button next to the 'Total' field. Set the 'Aggregate' to 'Max', and click the green 'Apply' button.



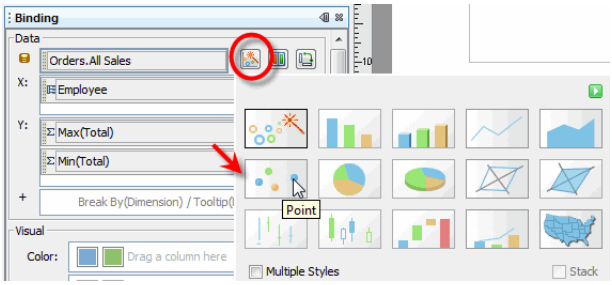
   c. From the 'All Sales' source, drag the 'Total' field (a second time) to the 'Y' region.

   d. Click the 'Edit Measure' button next to the second 'Total' field. Set the 'Aggregate' to 'Min', and click the green 'Apply' button.

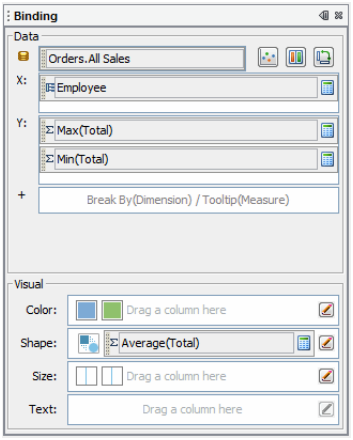
   e. From the 'All Sales' source, drag the 'Total' field (a third time) to the 'Shape' region of the 'Visual' area.

   f. Click the 'Edit Measure' button next to the third 'Total' field. Set the 'Aggregate' to 'Average', and click the green 'Apply' button.

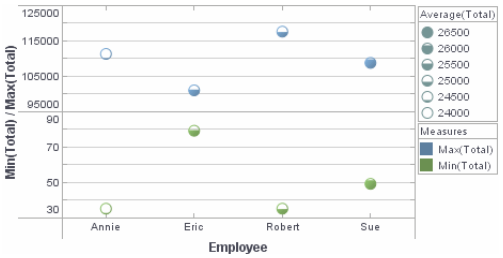
7. Click the 'Select Chart Style' button. Double-click to select the Point Style chart.



The Chart Editor should now appear as shown below:



8. Click the 'Preview' button the toolbar to preview the report or Viewsheet.



The chart shows maximum and minimum totals for each employee, and the chart shape-coding (interior fill) displays the average totals.

Note that the interior fill, 'Average(Total)', is the same for both datasets, so it is not needed in both locations. In the next steps, you will change the 'Min(Total)' markers to a solid red arrow-shape. You will also increase the size of the 'Max(Total)' markers so that the fill level is more visible. To make these changes, you will use Chart API functions.

```
Note: Chart API script which operates on the Chart's 'Egraph'
property should be placed in element-level script.
```

9. Open the Script Editor as follows:

    a. In a report, right-click on the chart and select ‘Script’ from the context menu.

    b. In a Viewsheet, right-click the chart and select ‘Properties’ from the context menu. Then select the **Script** tab.

10. Use script commands to create the required *StaticShapeFrame*, *StaticColorFrame*, and *StaticSizeFrame* objects.

```
// Create arrow-shaped markers:
var shpframe = new StaticShapeFrame(GShape.ARROWBAR);

// Create static red color:
var colframe = new
StaticColorFrame(java.awt.Color(0xFF0000));

// Create static size of 10 pixels:
var sizframe = new StaticSizeFrame(10);
```

11. Obtain a handle to each of the two datasets (element sets) by using the *EGraph.getElement(index)* method.

```
var elem0 = graph.getElement(0); // Max point element
var elem1 = graph.getElement(1); // Min point element
```

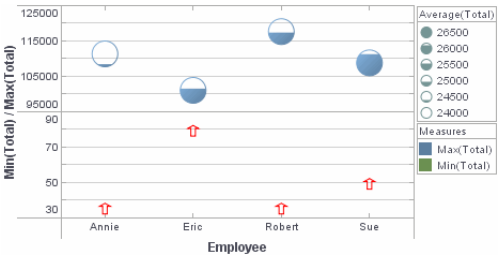
12. Assign the visual frames to the appropriate data elements using the element’s *GraphElement.setShapeFrame(frame)*, *GraphElement.setColorFrame(frame)*, and *GraphElement.setSizeFrame(frame)* properties.

```
elem1.shapeFrame = shpframe; // Min point element
elem1.colorFrame = colframe; // Min point element
elem0.sizeFrame = sizframe; // Max point element
```

13. Close the Script Editor, and preview the report or Viewsheet.

The complete script is shown below, along with the resulting graph.

```
var shpframe = new StaticShapeFrame(GShape.ARROWBAR);
var colframe = new
StaticColorFrame(java.awt.Color(0xFF0000));
var sizframe = new StaticSizeFrame(10);
var elem0 = graph.getElement(0); // Max point element
var elem1 = graph.getElement(1); // Min point element
elem1.shapeFrame = shpframe;
elem1.colorFrame = colframe;
elem0.sizeFrame = sizframe;
```



See Also

Creating a Chart Using API Functions, to create a chart from the ground-up using script.

Modifying a Chart Data Binding, to modify data binding information.

Appendix CT:*Chart Script Tutorial*, for examples of modifying chart elements.

Appendix CR:*Chart Script Reference*, for information on objects and attributes.

API Documentation for the `inetsoft.graph.*` packages.

CT.4 Creating a Chart Using API Functions

Previous sections explained how to modify the data binding and element properties of an existing chart. This section explains how to create a new chart (from the ground-up) using Chart API commands.

Note that charts created by script do not provide end-user interactivity features. Therefore, for script-based charts you should deselect 'Enable Ad Hoc Editing' in the 'Chart Properties' dialog box (**Advanced** tab).

Walkthrough

In this example, you will create a new chart, define the chart data, and display the data on the chart. Follow the steps below.

1. Add a new chart to the report or Viewsheet.

```
Note: Chart API script (which operates on the Chart's 'Egraph'
property) should be placed in element-level script.
```

2. Open the Script Editor as follows:

   a. In a report, right-click on the chart and select 'Script' from the context menu.

   b. In a Viewsheet, right-click the chart and select 'Properties' from the context menu. Then select the **Script** tab.

Tip: To reference chart data, use the *DataSet* object.

3. **Define the data**: Define the dataset for the chart using the Chart's *Data* property.

```
data = runQuery("sales by state");
```

For reports, you will generally obtain the data from a query, Worksheet, or data model by using `runQuery()`. In Viewsheets, you can additionally bind data from any available Data Block. You can also define your own dataset in script by assigning a JavaScript array to the `data` or `dataset` property. See *Binding Data to a Chart in Script* for more details.

EGraph is the global chart object, which includes all axes, legends, visual elements, etc.

4. **Create the Chart object**. Create a new graph using the *EGraph* constructor. Assign it to the Chart's *`graph`* property.

   ```
   graph = new EGraph();
   ```

5. **Create the chart data elements**. Pass the field names (column headers) to a *GraphElement* constructor. This creates the representational elements for the chart.

   ```
   var elem = new IntervalElement("State", "Sales");
   ```

   The *IntervalElement* is a particular type of *GraphElement* that creates "bars" or "intervals" as the representational elements. Other *GraphElements*, such as *PointElement* and *LineElement*, generate other kinds of chart types, such as scatter plots and line plots, respectively.

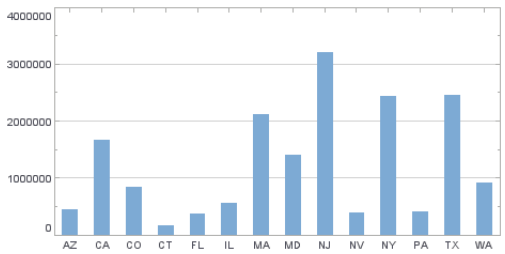
6. **Add the created elements to the Chart object**. Pass the *GraphElement* object to the Chart's *`EGraph.addElement(elem)`* method. This adds the "bar" elements to the existing Chart object.

   ```
   graph.addElement(elem);
   ```

7. Close the Script Editor and preview the report or Viewsheet.

These are the basic operations required to create a new chart (from the ground-up) using the Chart API. The complete script is shown below, along with the resulting chart.

```
data = runQuery("sales by state");
graph = new EGraph();
var elem = new IntervalElement("State", "Sales");
graph.addElement(elem);
```



See Also

Modifying a Chart Element using API Functions, to modify an existing chart.

Modifying a Chart Data Binding, to modify data binding information.

Appendix CT:*Chart Script Tutorial*, for examples of modifying chart elements.

Appendix CR:*Chart Script Reference*, for information on objects and attributes.

API Documentation for the `inetsoft.graph.*` packages.

CT.5 Binding Data to a Chart in Script

There are several ways to bind data to a chart using script. The following sections present the various approaches.

CT.5.1 Bind Chart Using Chart Editor (Report/Viewsheet)

The easiest way to bind data to a chart is to use the Chart Editor. After you have bound data to the chart using the Chart Editor, you can proceed to modify the chart using script operations. You can see an example of this hybrid approach in *Modifying a Chart Element using API Functions*.

CT.5.2 Bind Chart to Query or Worksheet (Report/Viewsheet)

`Use runQuery() only in element-level script.`

To bind data to a chart from an arbitrary query or Data Worksheet in element-level script, use the `runQuery()` function and assign the result to the chart's `data` or `dataset` property.

```
Chart1.data = runQuery("sales by state");
graph = new EGraph();
var elem = new IntervalElement("State", "Sales");
graph.addElement(elem);
```

You can see an example of this approach in *Creating a Chart Using API Functions*.

See Also

Modifying a Chart Data Binding, to configure the data binding of a chart.

Running a Query from Script in *Advanced Dashboard Scripting* details about the syntax of `runQuery()`.

Running a Query from Script in *Report Scripting* for details about the syntax of `runQuery()`.

CT.5.3 Bind Chart to Query or Worksheet (Report Only)

`Use the report 'query' property only in onLoad script.`

To bind data to a chart from an arbitrary query or Data Worksheet in a report *onLoad* script, use the chart's `query` property.

```
Chart1.query = 'sales by state';
Chart1.bindingInfo.xFields = [["State",Chart.STRING]];
Chart1.bindingInfo.yFields = [["Sales",Chart.NUMBER]];
```

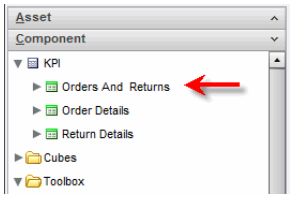
You can see an example of this approach in *Modifying a Chart Data Binding*.

To read Viewsheet data, use the Viewsheet's *Data* property.

CT.5.4 Bind Chart to Data Block (Viewsheet Only)

To bind data to one of the Data Blocks available to the Viewsheet, use the chart's `query` property (in component-level or onRefresh script).

For example, assume that 'Orders And Returns' is the name of a Data Block.



The following component-level or onRefresh script binds this Data Block to the chart.

```
Chart1.query = "Orders And Returns";
Chart1.bindingInfo.xFields = [["Company",Chart.STRING]];
Chart1.bindingInfo.yFields = [["Total",Chart.NUMBER]];
```

You can see an example of this approach in *Modifying a Chart Data Binding*.

CT.5.5 Bind Chart to JavaScript Array (Report/Viewsheet)

`Use array assignment only in element-level script.`

You can define a chart dataset entirely in element-level script by setting a JavaScript array as the `data` or `dataset` property.

```
Chart1.data = [["State","Quantity"],["NJ",200],["NY",300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
graph.addElement(elem);
```

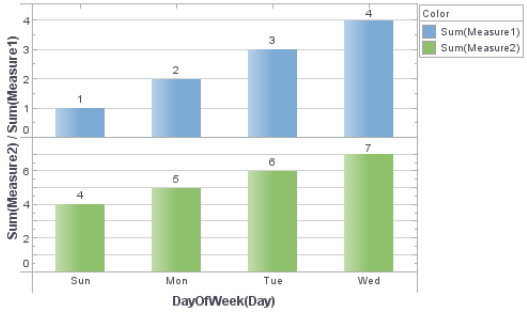
You can see an example of this approach in *Changing Scaling for Chart Axes*.

CT.6 Accessing Chart Data in Script

You can access Chart data in scripts by referencing the Chart's `data` or `table` property. The `table` object contains the aggregate values displayed on the chart (the same values shown by the 'Show Data' button in a Viewsheet Chart). The `table` object is accessible as a two-dimensional

array, where each column represents a distinct dataset (measure). The first column of the array, `table[i][0]`, contains the X-axis labels. The first row of the array, `table[0][i]`, contains the dataset or measure titles.

For example, consider a chart with two aggregated datasets (measures), as shown below:



The `table` object provides access to these aggregate values as the following array:

DayOfWeek(Day)	Sum(Measure1)	Sum(Measure2)
Sun	1.00	4.00
Mon	2.00	5.00
Tue	3.00	6.00
Wed	4.00	7.00

```
table[0][0] = 'DayOfWeek(Day)'
table[0][1] = 'Sum(Measure1)'
table[0][2] = 'Sum(Measure2)'
table[1][0] = 'Sun'
table[1][1] = 1
table[1][2] = 4
table[2][0] = 'Mon'
table[2][1] = 2
table[2][2] = 5
etc.
```

You can index the datasets (columns) numerically, e.g., `table[1][2]`, or by using the dataset name, e.g., `table[1]['Sum(Measure2)'])`. You can also use more complex formula table syntax; see *Data* in Appendix CR:*Chart Script Reference*.

In a Viewsheet chart, the `data` property and `table` property are identical. In a report chart, the `data` property represents the chart data prior to grouping and aggregation (i.e., the raw data).

Example: Accessing Chart Data

In this example, add a script to display data values on the chart only if the measure falls below a certain threshold.

1. Create a new report or Viewsheet. (For a Viewsheet, select the ‘Sales Explore’ Worksheet as the data source.)

2. Add a new Chart element to the report or Viewsheet. (For a Viewsheet, click the ‘Edit’ button on the Chart to open the Chart Editor.)

3. Bind the ‘Category’ field of the ‘Sales Explore’ Worksheet to the chart’s X-axis.

4. Bind both ‘Total’ and ‘Quantity Purchased’ fields of the ‘Sales Explore’ Worksheet to the chart’s Y-axis. This creates a facet chart with two sets of axes.

5. Resize the chart to show all data.

6. Open the Chart Script Editor:

   a. For a report, right-click on the chart and select ‘Script’ from the context menu. This opens the Script Editor.

   b. For a Viewsheet, right-click on the chart and select ‘Properties’ from the context menu. This opens the ‘Chart Properties’ dialog box. Select the **Script** tab

7. Add the following script:

```
var threshold = 5000;

// Step through the rows of chart data with index i
for (var i = 1; i < table.length; i++) {

  // Obtain the ith value of 'Category' and 'Quantity'
  var Xvalue = table[i][0];
  var Yvalue = table[i]['Sum(Quantity Purchased)'];

  // test the value of Quantity against the threshold
  if(Yvalue < threshold) {
    // Create the label object
    var form = new LabelForm();

    // Set the label to appear only on Quantity axes
    form.setMeasure('Sum(Quantity Purchased)')

    // Set the label text
    form.setLabel(Yvalue);

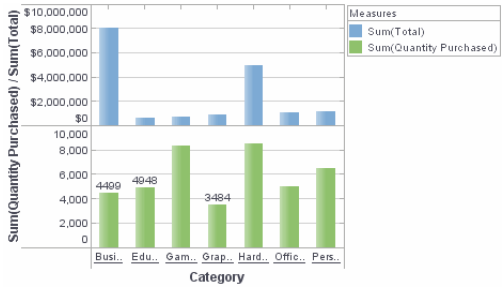
    // Set the label position and alignment
    form.setValues([Xvalue,Yvalue]);
    form.setAlignmentX(Chart.CENTER_ALIGNMENT);

    // Add the label to the graph
    graph.addForm(form)
  }
}
```

8. Press ‘OK’ to close the Script Editor.

9. Preview the report or Viewsheet.

Observe that values are shown for groups that have totals falling below the threshold of 5000.



¢

See Also

Data, in Appendix CR:*Chart Script Reference*, for additional information.

CT.7 Changing Chart Scaling

Some objects, such as *IntervalElement*, create an implicit *Scale*.

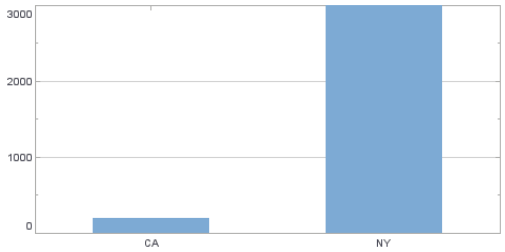
A *Scale* object determines how abstract data values are mapped to physical chart representations such as position, color, shape, etc. The *EGraph* object and the *VisualFrame* object both map data values to physical representations, so both of these objects require a *Scale*.

CT.7.1 Changing Scaling for Chart Axes

To change the scaling of chart axes, simply assign a new *Scale* to the Chart object. For example, consider the following chart:

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

```
dataset = [["State","Quantity"], ["CA",200], ["NY",3000]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
graph.addElement(elem);
```



Note that in this chart the *IntervalElement* object implicitly defines a linear Y-axis scale. However, the data values on the chart are widely different in magnitude, which suggests that a log scale might be more appropriate.

To change the Y-axis to use a log scale, follow these steps:

1. Define the desired *Scale* object explicitly. In this case, create a *LogScale* based on the 'Quantity' field.

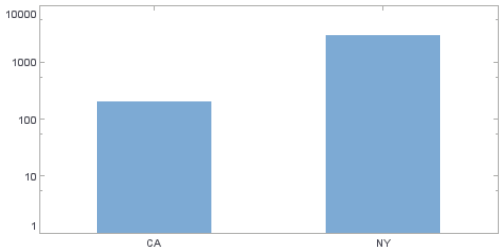
   ```
   var scale = new LogScale("Quantity");
   ```

2. Assign the new scale to the Y-axis of the chart object using the `setScale()` method.

   ```
   graph.setScale("Quantity",scale);
   ```

The complete script with the new chart scaling looks like this:

```
dataset = [["State","Quantity"], ["CA",200], ["NY",3000]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var scale = new LogScale("Quantity");
graph.addElement(elem);
graph.setScale("Quantity",scale);
```



Example: Changing Axis Range

In this example (for Viewsheets), you will dynamically modify the axis range to match the data displayed on the chart. In particular, you will adapt the lower limit on the Y-axis so that it is always 75% of the lowest data value shown on the chart.

Follow the steps below:

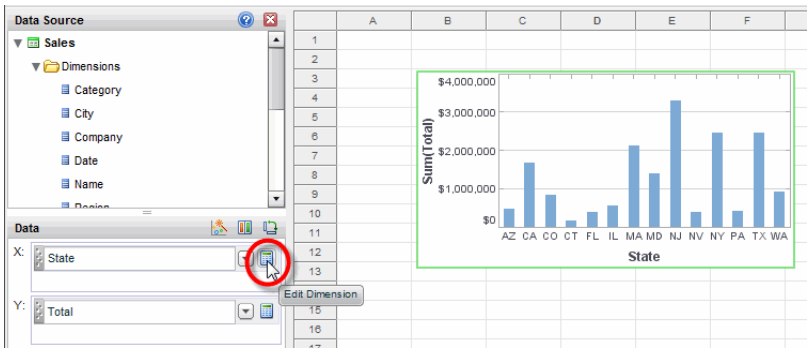
1. Create a new Viewsheet based on the 'Sales' > 'Sales Explore' Worksheet.

2. Add a Chart component to the Viewsheet, and click the center of the chart to open the Chart Editor.

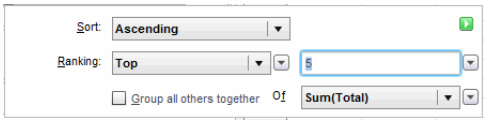
3. From the 'Dimensions' node of the 'Data Source' panel, drag the 'State' field to the 'X' region of the 'Data' panel.

4. From the ‘Measures’ node of the ‘Data Source’ panel, drag the ‘Total’ field to the ‘Y’ region of the ‘Data’ panel.

5. Click the ‘Edit Dimension’ button next to the ‘State’ field in the ‘Data’ panel.



6. From the ‘Ranking’ option in the ‘Edit Dimension’ panel, specify ‘Top 5 of Sum(Total)’. Click the ‘Apply’ button.



7. Right-click the Chart component on the Viewsheet grid, and select ‘Properties’ from the context menu. This opens the ‘Chart Properties’ dialog box.

Note: To access the data prior to chart aggregation, see *Accessing Worksheet Data* in *Dashboard Scripting*.

8. Select the **Script** tab, and enter the following script:

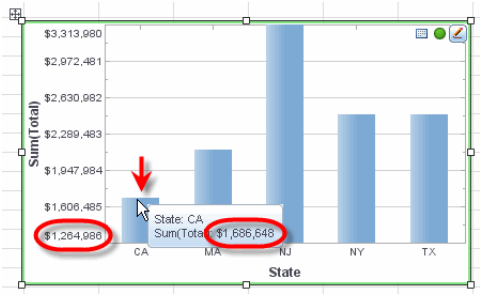
```
var dMin = 10000000; // Default minimum

// Get a handle to the chart's Y-axis Scale:
var yScale = graph.getCoordinate().getYScale();

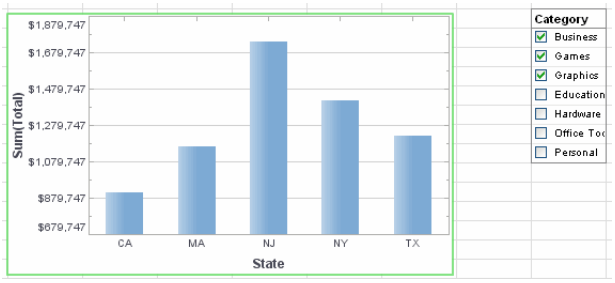
// Find the minimum Y-value in the chart data:
for (var i=0; i<dataset.getRowCount(); i++) {
  yVal = dataset.getData('Sum(Total)',i);
  if(yVal < dMin) {
    dMin = yVal;
  }
}

// Set Y-axis lower limit to .75 of minimum value:
yScale.setMin(.75*dMin);
```

9. Click the ‘OK’ button to close the ‘Chart Properties’ dialog box. Observe how the chart Y-axis updates so that the lower limit is 75% of the smallest chart value.



10. From the ‘Data Source’ panel of the Chart Editor, drag the ‘Category’ field onto the Viewsheet grid. This creates a ‘Category’ selection list.



11. Select different combinations of categories from the ‘Category’ selection list, and observe how the chart axis updates.

¢

See Also

Changing Axis Properties, for examples of other axis modifications.

CT.7.2Changing Scaling for a VisualFrame

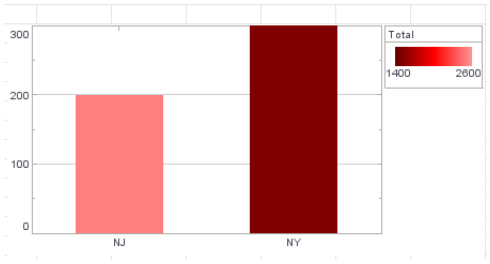
A *VisualFrame* object contains information about how data values are mapped to physical properties of chart elements. For example, a *BrightnessColorFrame* contains information about how data values in a field map to the brightness of corresponding chart elements. Mappings of this type require a *Scale*.

To change the scaling of a VisualFrame object, simply assign a new *Scale* to the *VisualFrame*. For example, consider the following chart:

```
Note: Script that modifies 'graph' should be placed at the element
level. See Adding Element-Level Script in Report Scripting and
Adding Component Script in Dashboard Scripting for more
information.

      var dataset =
      [["State","Quantity","Total"],["NJ",200,2500],["NY",300,150
      0]];
      graph = new EGraph();
```

```
var elem = new IntervalElement("State", "Quantity");
var frame = new BrightnessColorFrame();
frame.setField("Total");
frame.setColor(java.awt.Color(0xff0000));
elem.setColorFrame(frame);
graph.addElement(elem);
```



Note that in this chart a linear scale is defined implicitly by the *BrightnessColorFrame* object. The legend indicates that this default scale runs from 1400 to 2600. To change the scaling to run from 500 to 3000 instead, you must explicitly define a new *Scale* object.

Follow these steps:

1. Define the desired *Scale* object explicitly. In this case, create a *LinearScale* based on the 'Total' field.

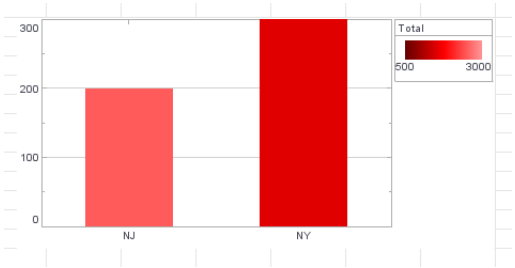
   ```
   var scale = new LinearScale("Total");
   scale.setFields("Total");
   ```

2. Set the minimum and maximum values of the new *Scale* object.

   ```
   scale.setMax(3000);
   scale.setMin(500);
   ```

3. Assign the new scale to the existing *VisualFrame* object.

   ```
   frame.setScale(scale);
   ```



The complete script with the new *VisualFrame* scaling looks like this:

```
dataset = [["State","Quantity","Total"],
["NJ",200,2500],["NY",300,1500]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
```

```
var frame = new BrightnessColorFrame();
frame.setField("Total");
frame.setColor(java.awt.Color(0xff0000));
var scale = new LinearScale("Total");
scale.setFields("Total");
scale.setMax(3000);
scale.setMin(500);
frame.setScale(scale);
elem.setColorFrame(frame);
graph.addElement(elem);
```

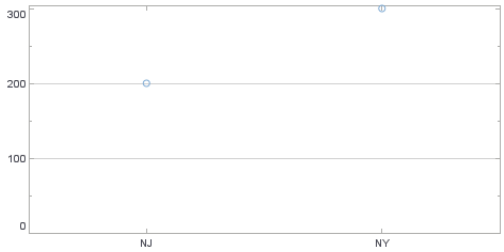
CT.8 Changing the Appearance of Chart Elements

You can change the static appearance of chart elements by using a static *VisualFrame*. For example, you can set static colors, sizes, and textures to enhance the aesthetic appearance of a chart.

Consider the script below:

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

```
dataset =
[["State","Quantity","Total"],["NJ",200,2500],["NY",300,150
0]];
graph = new EGraph();
var elem = new PointElement("State", "Quantity");
graph.addElement(elem);
```



This creates a basic point (scatter) chart displaying the dimensions ‘State’ and ‘Quantity’. However, the points are rather small and hard to see. To increase the size of the points and assign them a bolder color, use a *StaticColorFrame* and a *StaticSizeFrame*.

Follow these steps:

1. Create a new *StaticColorFrame* object, and specify a static color (red).

   ```
   var cframe = new StaticColorFrame();
   cframe.setColor(java.awt.Color(0xff0000)); // red
   ```

2. Create a new *StaticSizeFrame* object, and specify a static size.

```
var sframe = new StaticSizeFrame();
sframe.setSize(10);
```

3. Assign the *StaticColorFrame* and *StaticSizeFrame* objects to the *GraphElement* object.

```
elem.setColorFrame(cframe);
elem.setSizeFrame(sframe);
```



The complete script is shown below. The points are now large and red.

```
dataset =
[["State","Quantity","Total"],["NJ",200,2500],["NY",300,150
0]];
graph = new EGraph();
var elem = new PointElement("State", "Quantity");
var cframe = new StaticColorFrame();
cframe.setColor(java.awt.Color(0xff0000)); // red
var sframe = new StaticSizeFrame();
sframe.setSize(10);
elem.setColorFrame(cframe);
elem.setSizeFrame(sframe);
graph.addElement(elem);
```

Because these are *static VisualFrames*, the color and size are not keyed to the data. To represent data values using color, size, or other visual attributes, see *Representing Data with Shape, Color, Size*.

See Also

Representing Data with Shape, Color, Size, to use dynamic *VisualFrame* objects.

Adding Decorative Elements, for information on *GraphForm* objects.

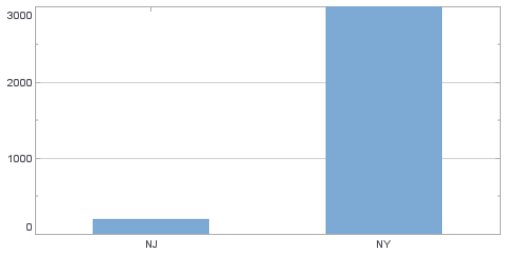
CT.9 Changing Axis Properties

To alter the appearance of chart axes, use the Chart's `setScale()` method to assign a new *Scale* object. For example, you can replace a linear scale with a logarithmic scale, show or hide tick marks, display axis labels at top or right, change the label font and color, etc.

Consider the following example:

```
Note: Script that modifies 'graph' should be placed at the element
level. See Adding Element-Level Script in Report Scripting and
Adding Component Script in Dashboard Scripting for more
information.
```

```
dataset = [["State","Quantity"], ["NJ",200], ["NY",3000]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
graph.addElement(elem);
```



This creates a basic bar chart displaying the dimensions ‘State’ and ‘Quantity’. Follow the steps below to experiment with modifying the chart’s axes:

1. Create a new logarithmic scale using the *LogScale* object, specifying ‘Quantity’ as the field on which the scale is based.

   ```
   var logscale = new LogScale('Quantity');
   ```

2. Set the color of the Y-axis lines and gridline to blue, and make the gridlines dotted. To do this, create a new *AxisSpec* object, and assign it to the *Scale*.

   ```
   var yspec = new AxisSpec();
   yspec.setLineColor(java.awt.Color(0x0000ff));
   yspec.setGridColor(java.awt.Color(0x0000ff));
   yspec.setGridStyle(Chart.DOT_LINE);
   logscale.setAxisSpec(yspec);
   ```

3. Create a new *CategoricalScale* for the X-axis, specifying ‘State’ as the field on which the scale is based.

   ```
   var cscale = new CategoricalScale('State');
   ```

4. Remove the X-axis lines and tick marks. To do this, create a new *AxisSpec* object, and assign it to the *Scale*.

   ```
   var xspec = new AxisSpec();
   xspec.setLineVisible(false);
   xspec.setTickVisible(false);
   cscale.setAxisSpec(xspec);
   ```

5. Move the X-axis labels above the chart area, and increase their size. To do this, create a new *TextSpec* object, and assign it to the *AxisSpec*.

   ```
   var tspec = new TextSpec();
   tspec.setFont(java.awt.Font('Dialog',
                 java.awt.Font.BOLD, 14));
   xspec.setTextSpec(tspec);
   ```

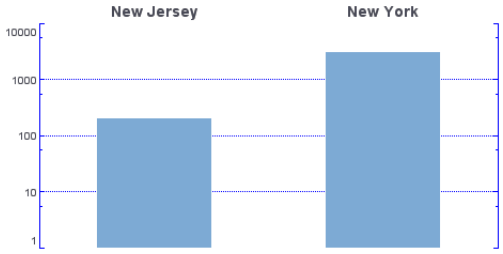
```
xspec.setAxisStyle(AxisSpec.AXIS_SINGLE2);
```

6. Create a new *TextFrame*, and specify new axis labels to replace the default labels ('NJ', 'NY') with the full state names. Assign the new *TextFrame* to the *AxisSpec* object.

```
var tframe = new DefaultTextFrame();
tframe.setText('NJ','New Jersey');
tframe.setText('NY','New York');
xspec.setTextFrame(tframe);
```

7. Assign the two *Scale* objects to the appropriate axes of the graph object.

```
graph.setScale('Quantity',logscale);
graph.setScale('State',cscale);
```



The complete script is shown below.

```
dataset = [["State","Quantity"], ["NJ",200], ["NY",3000]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var logscale = new LogScale('Quantity');
var yspec = new AxisSpec();
yspec.setLineColor(java.awt.Color(0x0000ff));
yspec.setGridColor(java.awt.Color(0x0000ff));
yspec.setGridStyle(Chart.DOT_LINE);
logscale.setAxisSpec(yspec);
var cscale = new CategoricalScale('State');
var xspec = new AxisSpec();
xspec.setLineVisible(false);
xspec.setTickVisible(false);
cscale.setAxisSpec(xspec);
var tspec = new TextSpec();
tspec.setFont(java.awt.Font('Dialog',java.awt.Font.BOLD,
14));
xspec.setTextSpec(tspec);
xspec.setAxisStyle(AxisSpec.AXIS_SINGLE2);
var tframe = new DefaultTextFrame();
tframe.setText('NJ','New Jersey');
tframe.setText('NY','New York');
xspec.setTextFrame(tframe);
graph.setScale('Quantity',logscale);
graph.setScale('State',cscale);
graph.addElement(elem);
```

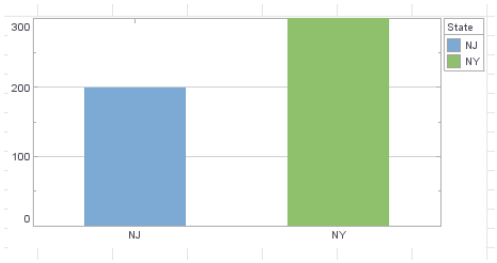
CT.10Changing Legend Properties

When you assign a *VisualFrame* to a chart element to visually code data, a corresponding legend is created automatically. You can change the appearance of this legend by editing the VisualFrame's *LegendSpec*.

Consider the following script:

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

```
dataset = [["State", "Quantity"], ["NJ",200], ["NY",300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var frame = new CategoricalColorFrame("State");
elem.setColorFrame(frame);
graph.addElement(elem);
```



This creates a basic bar chart displaying the dimensions 'State' and 'Quantity'. In this script, a *CategoricalColorFrame* distinguishes the different states by color. Follow the steps below to experiment with modifying the chart's legend:

1. Change the legend border to a blue dotted line. To do this, create a *LegendSpec* object and assign it to the *ColorFrame*.

   ```
   var legend = new LegendSpec();
   legend.setBorder(Chart.DOT_LINE);
   legend.setBorderColor(java.awt.Color(0x0000ff));
   frame.setLegendSpec(legend);
   ```

2. Change the legend title to say simply 'State', and make the text bold. To do this, create a *TextSpec* object and assign it to the *LegendSpec*.

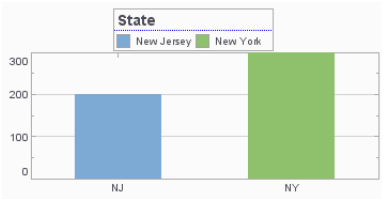
   ```
   var tspec = new TextSpec();
   tspec.setFont(java.awt.Font('Dialog',
                 java.awt.Font.BOLD, 14));
   legend.setTitleTextSpec(tspec);
   legend.setTitle('State');
   ```

3. Change the text inside the legend to display the full state names. To do this, create a *TextFrame* object and assign it to the *LegendSpec*.

```
var tframe = new DefaultTextFrame();
tframe.setText('NJ','New Jersey');
tframe.setText('NY','New York');
legend.setTextFrame(tframe);
```

4. Place the legend above the chart. Use the Graph's *legendLayout* property to do this.

```
graph.setLegendLayout(Chart.TOP);
```



The complete script is shown below.

```
dataset = [["State", "Quantity"], ["NJ",200], ["NY",300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var frame = new CategoricalColorFrame("State");
elem.setColorFrame(frame);

var legend = new LegendSpec();
legend.setBorder(Chart.DOT_LINE);
legend.setBorderColor(java.awt.Color(0x0000ff));
frame.setLegendSpec(legend);

var tspec = new TextSpec();
tspec.setFont(java.awt.Font('Dialog',java.awt.Font.BOLD,
14));
legend.setTitleTextSpec(tspec);
legend.setTitle('State');

var tframe = new DefaultTextFrame();
tframe.setText('NJ','New Jersey');
tframe.setText('NY','New York');
legend.setTextFrame(tframe);
graph.setLegendLayout(Chart.TOP);

graph.addElement(elem);
```

See Also

Representing Multiple Measures, for information on legends for multiple elements.

Representing Data with Shape, Color, Size, for more information on *VisualFrames*.

CT.11Changing Chart Labels

When you assign a *TextFrame* to a chart element to represent data textually, you can modify the labels by using *TextFrame.setText(value,text)*. The following example illustrates how you can construct custom labels for the chart using script. The labels display the names of the companies

represented on the chart together with each company's percentage of total sales.

Viewsheet

Follow the steps below if you are creating a Viewsheet. (For a report, skip to the next set of steps below.)

1. Create a new Viewsheet based on the 'All Sales' query. (See *Creating a Dashboard* for details.)

2. Drag a Chart component from the **Component** panel into the Viewsheet grid. Click the 'Edit' button on the chart to open the Chart Editor.

3. From the **Data Source** panel of the Chart Editor, drag the 'Company' field to the 'X' region of the **Data** panel.

4. From the **Data Source** panel of the Chart Editor, drag the 'Total' field to the 'Y' region of the **Data** panel.

5. Press the 'Edit Dimension' button next to the 'Company' field. From the 'Ranking' menu, select "Top 5 of Sum(Total)" and press the 'Apply' button.

6. Right-click the chart and select 'Properties' from the context menu. This opens the 'Chart Properties' dialog box.

7. Select the **Script** tab to access the Script Editor.

8. Enter the script shown below.

Report

Follow the steps below if you are creating a a report:

1. Create a new report. (See for details.)

2. Drag a Chart element from the **Toolbox** panel into the report. This adds the chart to the report and opens the Chart **Binding** panel.

3. In the **Data** panel, expand the 'Orders' node and expand the 'All Sales' node.

4. Drag the 'Company' field from the 'All Sales' query to the 'X' region of the **Binding** panel.

5. Drag the 'Total' field from the 'All Sales' query to the 'Y' region of the **Binding** panel.

6. Press the ‘Edit Dimension’ button next to the ‘Company’ field. From the ‘Ranking’ menu, select “Top 5 of Sum(Total)” and press the ‘Apply’ button.

7. Right-click the chart and select ‘Script’. This opens the Script Editor.

8. Enter the script shown below.

Viewsheet and Report

Enter the script shown below.

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

```
// Create new TextFrame based on 'Company' field:
var tframe = new DefaultTextFrame('Company');

// Get a handle to the graph element (bars):
var elem = graph.getElement(0);

// Assign the TextFrame to the element. This adds the
// category names above the bars (text-coding).
elem.setTextFrame(tframe)

// Compute the total amount of all companies:
var sumTotal = sum(table['Sum(Total)']);

// Loop through companies on chart:
for(i=1;i<table.length;i++) {

  // Get the original label (company):
  var oldLabel = table[i][0];

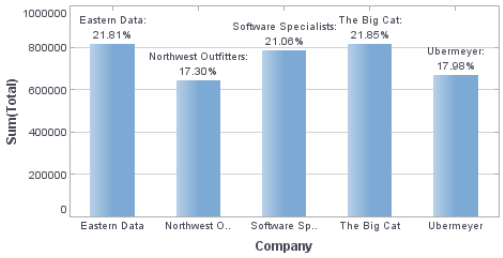
  // Compute the fractional value of corresponding 'Total':
  var barFraction = table[i][1]/sumTotal;

  // Format the fraction as a percent:
  var barPercent = formatNumber(barFraction,'##.00%');

  // Compose the new label, with form 'Company:Percent':
  var newLabel = oldLabel + ':\n' + barPercent;

  // Assign the new label in place of the old label:
  tframe.setText(oldLabel,newLabel);
}
```

The above script uses a customized *TextFrame* to display the company name together with the percentage of total represented by each company. The *TextFrame.setText(value,text)* function is the key to replacing one set of labels by a different set of labels.



In this example you created a completely new TextFrame to display the data, but you can use the same technique to modify an existing TextFrame. For example, to modify the existing X-axis labels, you would first get a handle to the existing X-axis TextFrame. For example.

```
var tframe =
graph.getCoordinate().getXScale().getAxisSpec().getTextFram
e();
```

Because this TextFrame is already associated with the X-axis labels, there is no need to assign the TextFrame to the axis. The only thing you need to do is to swap in the new labels. Here is the revised script in its entirety:

```
// Get a handle to the existing X-axis TextFrame:
var tframe =
graph.getCoordinate().getXScale().getAxisSpec().getTextFram
e();

// Get a handle to the graph element (bars):
var elem = graph.getElement(0);

// Compute the total amount of all companies:
var sumTotal = sum(table['Sum(Total)']);

// Loop through companies on chart:
for(i=1;i<table.length;i++) {

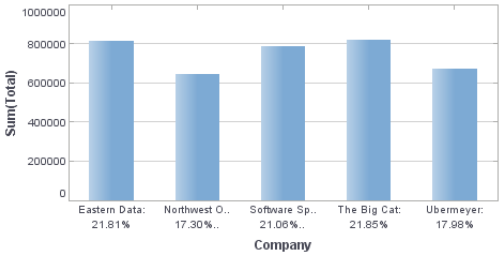
  // Get the original label (company):
  var oldLabel = table[i][0];

  // Compute the fractional value of corresponding 'Total':
  var barFraction = table[i][1]/sumTotal;

  // Format the fraction as a percent:
  var barPercent = formatNumber(barFraction,'##.00%');

  // Compose the new label, with form 'Company:Percent':
  var newLabel = oldLabel + ':\n' + barPercent;

  // Assign the new label in place of the old label:
  tframe.setText(oldLabel,newLabel);
}
```



CT.12Changing the Chart Coordinates

There are several different *Coordinate* objects, each of which creates a different kind of chart. The following sections discuss the different types of chart coordinate systems.

CT.12.1Rectangular Coordinates

The default coordinate set is rectangular coordinates, as defined by the *RectCoord* object. Rectangular coordinates represent two-dimensional data on horizontal (X) and vertical (Y) axes.

A *RectCoord* object is created automatically when you create a new Chart object. Therefore, if you plan to use rectangular coordinates on the chart, you *do not* need to explicitly specify a *RectCoord* object. You only need to explicitly create a *RectCoord* object when you define other types of coordinate systems. (See the following sections for more details.)

Assigning Rectangular Coordinates Automatically

The following example illustrates automatic creation of rectangular coordinates:

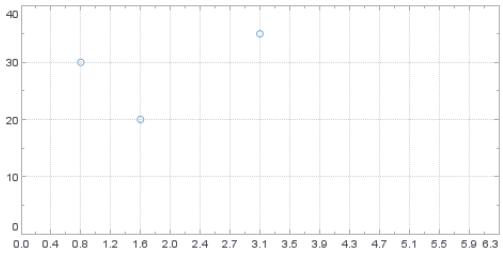
Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

```
dataset = [["Direction", "Score"],[(Math.PI/
2),20],[(Math.PI/4),30],[(Math.PI),35]];
graph = new EGraph();
var elem = new PointElement("Direction", "Score");
var xscale = new LinearScale("Direction");
var yscale = new LinearScale("Score");
yscale.setMin(0);
yscale.setMax(40);
var yaxis = new AxisSpec();
yaxis.setGridStyle(Chart.DOT_LINE);
yscale.setAxisSpec(yaxis);

xscale.setMin(0);
xscale.setMax(1.95*Math.PI);
xscale.setIncrement(Math.PI/8);
var xaxis = new AxisSpec();
var tspec = new TextSpec();
tspec.setFormat(new java.text.DecimalFormat("0.0"));
xaxis.setTextSpec(tspec);
```

```
xaxis.setGridStyle(Chart.DOT_LINE);
xscale.setAxisSpec(xaxis);

graph.setScale("Direction",xscale);
graph.setScale("Score",yscale);
graph.addElement(elem);
```



The script defines a *PointElement* style (scatter plot), and specifies a *LinearScale* for the 'Direction' field and 'Score' field. Note that the script does *not* explicitly create a *RectCoord* object. A rectangular coordinate system is created automatically, and this allows you to assign the specified scales directly to the Graph:

```
graph.setScale("Direction",xscale);
graph.setScale("Score",yscale);
```

Assigning Rectangular Coordinates Explicitly

You can explicitly define the *RectCoord* object, if needed. The script below is the same as the script in *Assigning Rectangular Coordinates Automatically*, but uses two scales to explicitly define a set of rectangular coordinates. These coordinates are then explicitly assigned to the chart.

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

```
dataset = [["Direction", "Score"],[(Math.PI/
2),20],[(Math.PI/4),30],[(Math.PI),35]];
graph = new EGraph();
var elem = new PointElement("Direction", "Score");
var xscale = new LinearScale("Direction");
var yscale = new LinearScale("Score");
yscale.setMin(0);
yscale.setMax(40);
var yaxis = new AxisSpec();
yaxis.setGridStyle(Chart.DOT_LINE);
yscale.setAxisSpec(yaxis);

xscale.setMin(0);
xscale.setMax(1.95*Math.PI);
xscale.setIncrement(Math.PI/8);
var xaxis = new AxisSpec();
var tspec = new TextSpec();
tspec.setFormat(new java.text.DecimalFormat("0.0"));
xaxis.setTextSpec(tspec);
xaxis.setGridStyle(Chart.DOT_LINE);
xscale.setAxisSpec(xaxis);
```

```
graph.setScale("Direction",xscale);
graph.setScale("Score",yscale);
var rect = new RectCoord(xscale,yscale);
graph.setCoordinate(rect);
graph.addElement(elem);
```

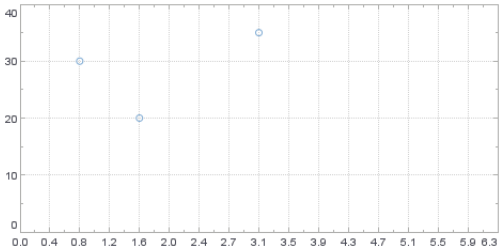
The resulting chart is the same as in the previous case.

CT.12.2Polar Coordinates

Polar coordinates represent data in terms of an angle and magnitude (radius). They are defined by the *PolarCoord* object, which accepts a *RectCoord* object as input. The following sections explain how to create and modify polar coordinates. They also demonstrate a common use of polar coordinates, the pie chart.

Converting Rectangular to Polar Coordinates

To use polar coordinates, you first need to create a set of rectangular coordinates. Consider the script from the previous section, *Rectangular Coordinates*:



Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

```
dataset = [["Direction", "Score"],[(Math.PI/
2),20],[(Math.PI/4),30],[(Math.PI),35]];
graph = new EGraph();
var elem = new PointElement("Direction", "Score");
var xscale = new LinearScale("Direction");
var yscale = new LinearScale("Score");
yscale.setMin(0);
yscale.setMax(40);
var yaxis = new AxisSpec();
yaxis.setGridStyle(Chart.DOT_LINE);
yscale.setAxisSpec(yaxis);

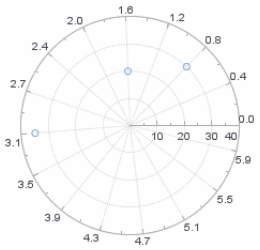
xscale.setMin(0);
xscale.setMax(1.95*Math.PI);
xscale.setIncrement(Math.PI/8);
var xaxis = new AxisSpec();
var tspec = new TextSpec();
tspec.setFormat(new java.text.DecimalFormat("0.0"));
xaxis.setTextSpec(tspec);
xaxis.setGridStyle(Chart.DOT_LINE);
xscale.setAxisSpec(xaxis);
```

```
graph.setScale("Direction",xscale);
graph.setScale("Score",yscale);
var rect = new RectCoord(xscale,yscale);
graph.setCoordinate(rect);
graph.addElement(elem);
```

Note that this script explicitly creates a set of rectangular coordinates by calling the *RectCoord* constructor. To convert the chart to a polar coordinate system, use this *RectCoord* object to create a new *PolarCoord* object:

```
var polar = new PolarCoord(rect);
```

Assign this new coordinate set to the chart to obtain the desired polar representation. The modified script is shown below:




```
dataset = [["Direction", "Score"],[(Math.PI/
2),20],[(Math.PI/4),30],[(Math.PI),35]];
graph = new EGraph();
var elem = new PointElement("Direction", "Score");
var xscale = new LinearScale("Direction");
var yscale = new LinearScale("Score");
yscale.setMin(0);
yscale.setMax(40);
var yaxis = new AxisSpec();
yaxis.setGridStyle(Chart.DOT_LINE);
yscale.setAxisSpec(yaxis);

xscale.setMin(0);
xscale.setMax(1.95*Math.PI);
xscale.setIncrement(Math.PI/8);
var xaxis = new AxisSpec();
var tspec = new TextSpec();
tspec.setFormat(new java.text.DecimalFormat("0.0"));
xaxis.setTextSpec(tspec);
xaxis.setGridStyle(Chart.DOT_LINE);
xscale.setAxisSpec(xaxis);

graph.setScale("Direction",xscale);
graph.setScale("Score",yscale);
var rect = new RectCoord(xscale,yscale);
var polar = new PolarCoord(rect);
graph.setCoordinate(polar);
graph.addElement(elem);
```

By default, the rectangular coordinate's X-axis is mapped to the polar coordinate's angle, and the rectangular coordinate's Y-axis is mapped to the

polar coordinate's magnitude (radius). To reverse a mapping, use the *Coordinate* object's `transpose()` method.

Tailoring a Coordinate Mapping

When converting rectangular to polar coordinates, you can choose to map only one of the dimensions. Use the *PolarCoord* Type property to do this.

Mapping just a single dimension is useful for pie charts. See *Example: Pie Chart*.

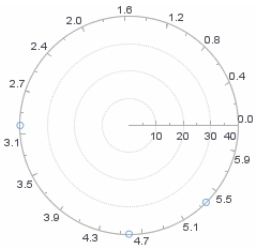
The example below is the same as that in *Converting Rectangular to Polar Coordinates*, except that here only the angle dimension is mapped:

```
Note: Script that modifies 'graph' should be placed at the element
level. See Adding Element-Level Script in Report Scripting and
Adding Component Script in Dashboard Scripting for more
information.
```

```
dataset = [["Direction", "Score"],[(Math.PI/
2),20],[(Math.PI/4),30],[(Math.PI),35]];
graph = new EGraph();
var elem = new PointElement("Direction", "Score");
var xscale = new LinearScale("Direction");
var yscale = new LinearScale("Score");
yscale.setMin(0);
yscale.setMax(40);
var yaxis = new AxisSpec();
yaxis.setGridStyle(Chart.DOT_LINE);
yscale.setAxisSpec(yaxis);

xscale.setMin(0);
xscale.setMax(1.95*Math.PI);
xscale.setIncrement(Math.PI/8);
var xaxis = new AxisSpec();
var tspec = new TextSpec();
tspec.setFormat(new java.text.DecimalFormat("0.0"));
xaxis.setTextSpec(tspec);
xaxis.setGridStyle(Chart.DOT_LINE);
xscale.setAxisSpec(xaxis);

graph.setScale("Direction",xscale);
graph.setScale("Score",yscale);
var rect = new RectCoord(xscale,yscale);
var polar = new PolarCoord(rect);
polar.setType(PolarCoord.THETA);
graph.setCoordinate(polar);
graph.addElement(elem);
```



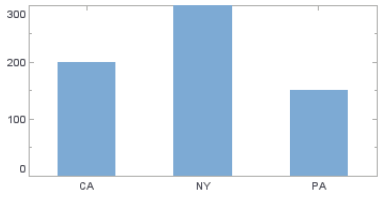
The result is that all points have the same magnitude, with variation only along the angle dimension. To reverse a mapping, use the *Coordinate* object's `transpose()` method.

Example: Pie Chart

You can think of a pie chart as a stacked bar chart with just one bar, displayed in polar coordinates. To see this, consider the script below, which creates a simple bar chart.

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

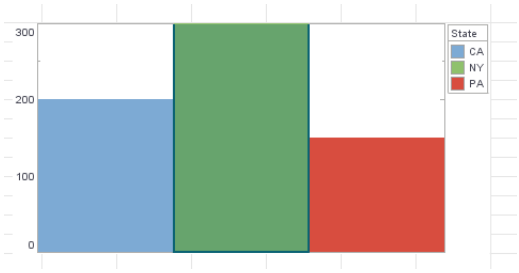
```
dataset = [["State", "Revenue"], ["CA",
200],["NY",300],["PA",150]];
graph = new EGraph();
var elem = new IntervalElement("State","Revenue");
var xscale = new CategoricalScale("State")
var yscale = new LinearScale("Revenue");
var rect = new RectCoord(xscale, yscale);
graph.setCoordinate(rect);
graph.addElement(elem);
```



To create a pie chart from this data, just convert the chart to a stacked bar, and then convert to polar coordinates. Follow the steps below:

1. Instead of using different X-axis positions to distinguish the 'State' data, distinguish the states using a *ColorFrame* object.

```
dataset = [["State", "Revenue"], ["CA",
200],["NY",300],["PA",150]];
graph = new EGraph();
var elem = new IntervalElement(null,"Revenue");
var xscale = new CategoricalScale("State")
var yscale = new LinearScale("Revenue");
var rect = new RectCoord(null, yscale);
var cframe = new CategoricalColorFrame("State");
elem.setColorFrame(cframe);
graph.setCoordinate(rect);
graph.addElement(elem);
```



Note that a 'null' value is assigned for the (unused) X-dimension when creating the *IntervalElement* and *RectCoord* objects.

2. Convert the chart into a stacked bar chart with just one bar by using the *GraphElement*'s collisionModifier property. Create a *StackRange* object to make sure there is enough room for the stacked elements.

```
elem.setCollisionModifier(GraphElement.STACK_SYMMETRIC);
yscale.setScaleRange(new StackRange());
```

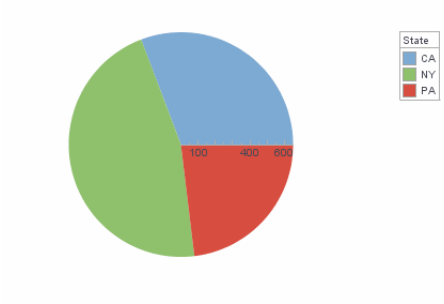


3. Create polar coordinates from the existing rectangular coordinates. By default, this maps the non-null coordinate (Y-axis) to the polar coordinate's magnitude dimension. Specify that it should be mapped to the angle dimension instead.

```
var polar = new PolarCoord(rect);
polar.setType(PolarCoord.THETA);
```

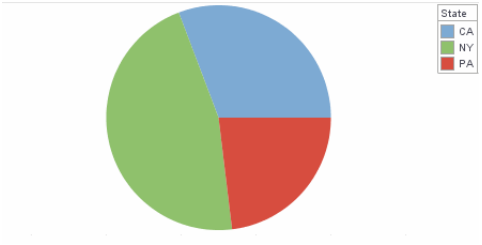
4. Use the Chart's `setCoordinate()` method to apply the new polar coordinates.

```
graph.setCoordinate(polar);
```



5. Remove the axis lines and labels. To do this, create an *AxisSpec* object, and assign it to the *Scale*.

```
var yspec = new AxisSpec();
yspec.setLabelVisible(false);
yspec.setLineVisible(false);
yspec.setTickVisible(false);
yscale.setAxisSpec(yspec);
```

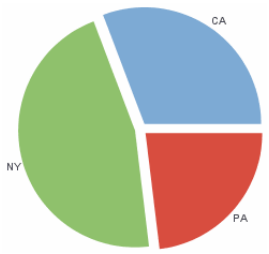


6. Add the state names to the individual slices and hide the legend. To do this, create a new *TextFrame* object based on the "State" field, and assign it to the *GraphElement*. To hide the legend, create a new *LegendSpec* object, and assign it to the *ColorFrame*.

```
var tframe = new DefaultTextFrame("State");
elem.setTextFrame(tframe);
var legend = new LegendSpec();
legend.setVisible(false);
cframe.setLegendSpec(legend);
```

7. Explode the slices for better appearance.

```
elem.setHint(GraphElement.HINT_EXPLODED,'true');
```



The final script for the pie chart is shown below.

```
dataset = [["State", "Revenue"], ["CA",
200],["NY",300],["PA",150]];
graph = new EGraph();
var elem = new IntervalElement(null,"Revenue");
var xscale = new CategoricalScale("State")
var yscale = new LinearScale("Revenue");
var rect = new RectCoord(null, yscale);
var cframe = new CategoricalColorFrame("State");
elem.setColorFrame(cframe);
elem.setCollisionModifier(GraphElement.STACK_SYMMETRIC);
yscale.setScaleRange(new StackRange());
var polar = new PolarCoord(rect);
polar.setType(PolarCoord.THETA);
var yspec = new AxisSpec();
yspec.setLabelVisible(false);
yspec.setLineVisible(false);
yspec.setTickVisible(false);
yscale.setAxisSpec(yspec);
var tframe = new DefaultTextFrame("State");
elem.setTextFrame(tframe);
var legend = new LegendSpec();
legend.setVisible(false);
```

```
cframe.setLegendSpec(legend);
elem.setHint(GraphElement.HINT_EXPLODED,'true');
graph.setCoordinate(polar);
graph.addElement(elem);
```

CT.12.3Parallel Coordinates

Parallel coordinates display multiple dimensions as parallel axes, rather than orthogonal axes, as for rectangular coordinates. Parallel coordinates are defined by the *ParallelCoord* object, which accepts a set of *Scale* objects as input.

To understand parallel coordinates, consider the following data set, which contains scores for three students over three consecutive tests.

Test 1	Test 2	Test 3	Name
200	175	50	Joe
800	1000	300	Jane
10	15	20	Fred

By plotting this data on three parallel coordinates (‘Test 1’, ‘Test 2’, ‘Test 3’), you can visualize trends across the different tests. To create this chart, follow the steps below:

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

1. Define the data set and chart objects.

```
dataset =
[["Test1","Test2","Test3","Name"],[100,80,20,'Joe'],[75,50,
40,'Jane'],[50,30,80,'Fred']];
graph = new EGraph();
```

2. Create a new *LineElement* object to define a line-style chart, and assign its dimensions.

```
var elem = new LineElement();
elem.addDim("Test1");
elem.addDim("Test2");
elem.addDim("Test3");
```

3. Define the scales used for the three axes. (In this case the scales are the same for all three.)

```
var scale1 = new LinearScale("Test1");
var scale2 = new LinearScale("Test2");
var scale3 = new LinearScale("Test3");
scale1.setMax(100);
scale2.setMax(100);
scale3.setMax(100);
scale1.setMin(0);
scale2.setMin(0);
```

```
scale3.setMin(0);
```

4. Create the new *ParallelCoord* object using the defined scales.

```
var coord = new ParallelCoord(scale1,scale2,scale3);
```

5. Use a *CategoricalColorFrame* to distinguish the three students. Assign the frame to the line element.

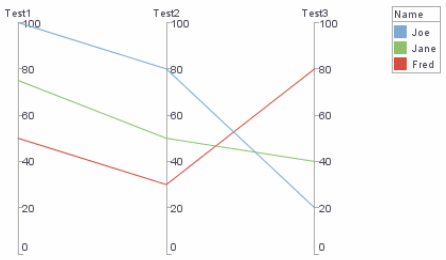
```
var frame = new CategoricalColorFrame("Name");
elem.setColorFrame(frame);
```

6. Assign the parallel coordinate system to the chart.

```
graph.addElement(elem);
graph.setCoordinate(coord);
```

The complete script is shown below:

```
dataset =
[["Test1","Test2","Test3","Name"],[100,80,20,'Joe'],[75,50,
40,'Jane'],[50,30,80,'Fred']];
graph = new EGraph();
var elem = new LineElement();
elem.addDim("Test1");
elem.addDim("Test2");
elem.addDim("Test3");
var scale1 = new LinearScale("Test1");
var scale2 = new LinearScale("Test2");
var scale3 = new LinearScale("Test3");
scale1.setMax(100);
scale2.setMax(100);
scale3.setMax(100);
scale1.setMin(0);
scale2.setMin(0);
scale3.setMin(0);
var coord = new ParallelCoord(scale1,scale2,scale3);
var frame = new CategoricalColorFrame("Name");
elem.setColorFrame(frame);
graph.addElement(elem);
graph.setCoordinate(coord);
```



CT.12.4Facet Coordinates

The *FacetCoord* object contains a set of inner and outer coordinates on which multidimensional data can be represented as nested charts. To create

a *FacetCoord* object, pass a pair of *RectCoord* objects to the *FacetCoord* constructor:

```
var rect = new FacetCoord(outerCoord,innerCoord);
```

To understand facet coordinates, consider the following data set.

State	Product	Name	Priority
NJ	P1	Joe	2
NJ	P2	Sam	3
NY	P1	Jane	4
NJ	P1	Sam	1
NJ	P2	Joe	10
NY	P1	Sam	10

Because there are four different dimensions, there are several ways to look at the data. For example, you may want to plot 'Priority' vs. 'Name', and also break this down by 'Product' and 'State'. To construct a facet chart to do this, follow the steps below:

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

1. Define the data and the Chart objects, and create a new *IntervalElement* (bar chart).

```
dataset = [["State", "Product", "Name", "Priority"],["NJ",
"P1", "Joe", 2],["NJ", "P2", "Sam", 3],["NY", "P1", "Jane",
4],["NJ", "P1", "Sam", 1],["NJ", "P2", "Joe", 10],["NY",
"P1", "Sam", 10]];
graph = new EGraph();
var elem = new IntervalElement("Name", "Priority");
```

2. Create a *Scale* for each of the dimensions. All dimensions are categorical except for 'Priority'.

```
var state = new CategoricalScale("State");
var name = new CategoricalScale("Name");
var product = new CategoricalScale("Product");
var priority = new LinearScale("Priority");
```

3. Define two sets of rectangular coordinates, one for the outer coordinates ('Product' vs. 'State') and one for the inner coordinates ('Priority' vs. 'Name').

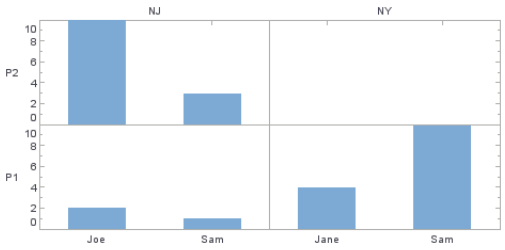
```
var inner = new RectCoord(name, priority);
var outer = new RectCoord(state, product);
```

4. Create the facet coordinates based on the outer and inner rectangular coordinates defined above.

```
var coord = new FacetCoord(outer,inner);
```

5. Assign the coordinates and the bar elements to the chart.

```
graph.setCoordinate(coord);
graph.addElement(elem);
```



The resulting chart displays an outer grid based on the outer coordinates ('State' and 'Product'). Within each cell of the outer grid, the chart displays the corresponding inner coordinates ('Name' and 'Priority').

The complete script is provided below.

```
dataset = [["State", "Product", "Name", "Priority"],["NJ",
"P1", "Joe", 2],["NJ", "P2", "Sam", 3],["NY", "P1", "Jane",
4],["NJ", "P1", "Sam", 1],["NJ", "P2", "Joe", 10],["NY",
"P1", "Sam", 10]];
graph = new EGraph();
var elem = new IntervalElement("Name", "Priority");
var state = new CategoricalScale("State");
var name = new CategoricalScale("Name");
var product = new CategoricalScale("Product");
var priority = new LinearScale("Priority");
var inner = new RectCoord(name, priority);
var outer = new RectCoord(state, product);
var coord = new FacetCoord(outer,inner);
graph.setCoordinate(coord);
graph.addElement(elem)
```

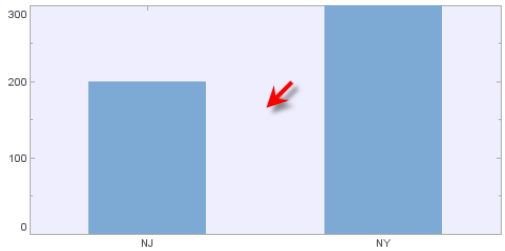
CT.12.5Setting a Coordinate Background

You can assign a background color or background image to the coordinate area (plot area) of a chart by defining a *PlotSpec* object. The following example illustrates how to set a blue background for a chart:

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

```
dataset = [["State", "Quantity"], ["NJ", 200], ["NY", 300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var sscale = new CategoricalScale("State");
var qscale = new LinearScale("Quantity");
var coord = new RectCoord(sscale,qscale);
var spec = new PlotSpec();
spec.setBackground(java.awt.Color(0xEEEEFF));
```

```
coord.setPlotSpec(spec);
graph.setCoordinate(coord);
graph.addElement(elem);
```



Use the `setBackgroundImage()` function to display an image as the plot background. The following example uses a static Google map image as a background by aligning the map coordinates with the chart coordinates.

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

```
// Define latitude and longitude values for landmarks:
dataset =
[["Latitude","Longitude","PlaceName"],[40.8516051126306,-
73.95223617553711,' GW Bridge '],[40.76292614285948,-
74.00982856750488,' Lincoln Tunnel '],[40.72755146730012,-
74.02107238769531,' Holland Tunnel ']];
graph = new EGraph();

// Define chart elements and basic point appearance:
var elem = new PointElement("Longitude","Latitude");
var tframe = new DefaultTextFrame("PlaceName");
var sframe = new StaticShapeFrame();
var cframe = new StaticColorFrame();
cframe.setColor(java.awt.Color(0x0000000));
sframe.setShape(GShape.FILLED_CIRCLE);

// Define appearance of text labels:
var tspec = new TextSpec();
tspec.setBackground(java.awt.Color(0x0000000))
tspec.setFont(java.awt.Font('Trebuchet',java.awt.Font.BOLD,
11));
tspec.setColor(java.awt.Color(0xffff00));

// Configure a background image using PlotSpec:
var pspec = new PlotSpec();
pspec.setLockAspect(true);
var logo = getImage("http://maps.google.com/maps/api/
staticmap?center=40.7857,-
73.9819&zoom=11&size=400x400&sensor=false");
pspec.setBackgroundImage(logo);
pspec.setYMax(40.8902)   // YMax = high latitude
pspec.setYMin(40.6822)   // YMin = low latitude
pspec.setXMax(-73.84529) // XMax = high longitude
pspec.setXMin(-74.1206)  // XMin = low longitude

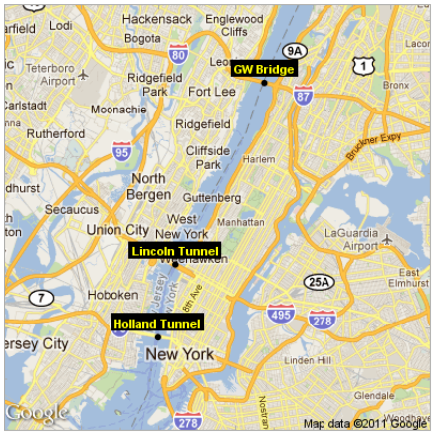
// Set chart scales to match image coordinates:
var latscale = new LinearScale("Latitude");
var lonscale = new LinearScale("Longitude");
```

```
var aspec = new AxisSpec();
aspec.setLabelVisible(false);
latscale.setAxisSpec(aspec);
lonscale.setAxisSpec(aspec);
latscale.setScaleOption(0);
lonscale.setScaleOption(0);
latscale.setMax(pspec.getYMax());  // match YMax
latscale.setMin(pspec.getYMin());  // match YMin
lonscale.setMax(pspec.getXMax());  // match XMax
lonscale.setMin(pspec.getXMin());  // match XMin
var coord = new RectCoord(lonscale,latscale);
coord.setPlotSpec(pspec);

// Assign visual frames to chart elements:
elem.setTextFrame(tframe);
elem.setTextSpec(tspec);
elem.setShapeFrame(sframe);
elem.setColorFrame(cframe);
elem.setHint(GraphElement.HINT_ALPHA,1);

// Assign coordinates. Generate chart with appropriate size:
graph.setCoordinate(coord);
graph.addElement(elem);

// For a report, set the chart element size as desired:
//Chart1.size = [5.6,5.6];
```



Note that the limits of the chart scales (`latscale.setMax`, `latscale.setMin`, `lonscale.setMax`, `lonscale.setMin`) have been set to match the geographical boundaries of the map image. This allows the image to align correctly with the chart axes.

See Also

Appendix CR.4.9 , *PlotSpec*, for reference information.

background, in *Report Scripting*, to set a background for the report element.

Chart Properties, in *Advanced Report Design*, to set a background for the report element.

Color Tab, in *Advanced Dashboard Design*, to set a background for the Viewsheet element.

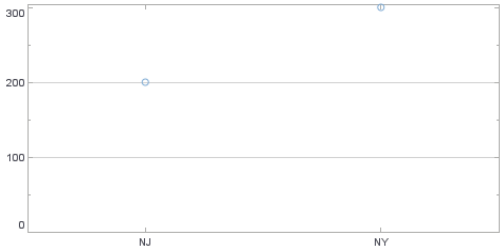
CT.13Representing Data with Shape, Color, Size

You can create a basic two-dimensional representation of data with just a *GraphElement* object. To represent additional dimensions by using other visual attributes of elements, create a *VisualFrame*.

To understand how to use a *VisualFrame*, consider the script below:

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

```
dataset =
[["State","Quantity","Total"],["NJ",200,2500],["NY",300,150
0]];
graph = new EGraph();
var elem = new PointElement("State", "Quantity");
graph.addElement(elem);
```



This creates a basic point (scatter) chart displaying the dimensions ‘State’ and ‘Quantity’. If you want to additionally represent the dimension ‘Total’ with the element *size*, use a VisualFrame such as the *LinearSizeFrame*. Follow these steps:

1. Create a new *LinearSizeFrame* object, and specify the field containing the data that will determine the element sizes.

   ```
   var frame = new LinearSizeFrame();
   frame.setField("Total");
   ```

A VisualFrame object such as LinearSize-Frame contains a mapping between data values and physical attributes. Therefore, you need a *Scale* to specify the mapping’s scaling.

2. Create a new *LinearScale* object, and assign the scale properties. (See Appendix CT.7, *Changing Chart Scaling*, for more information.)

   ```
   var scale = new LinearScale();
   scale.setFields("Total");
   scale.setMax(3000);
   scale.setMin(1000);
   ```

3. Assign the new *Scale* to the *VisualFrame* object:

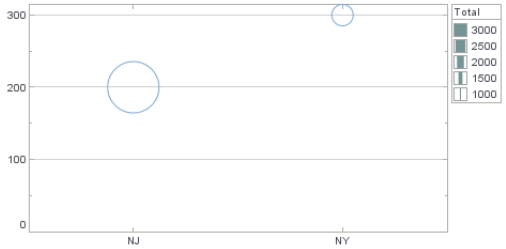
   ```
   frame.setScale(scale);
   ```

4. Assign the *VisualFrame* object to the *GraphElement* object:

```
elem.setSizeFrame(frame);
```

The point sizes now represent the data values contained in the ‘Total’ field. The complete script is shown below.

```
dataset =
[["State","Quantity","Total"],["NJ",200,2500],["NY",300,150
0]];
graph = new EGraph();
var elem = new PointElement("State", "Quantity");
var frame = new LinearSizeFrame();
frame.setField("Total");
var scale = new LinearScale();
scale.setFields("Total");
scale.setMax(3000);
scale.setMin(1000);
frame.setScale(scale);
elem.setSizeFrame(frame);
graph.addElement(elem);
```



See Also

Representing Multiple Measures, for information on adding multiple elements.

Changing the Appearance of Chart Elements, to use static *VisualFrame* objects.

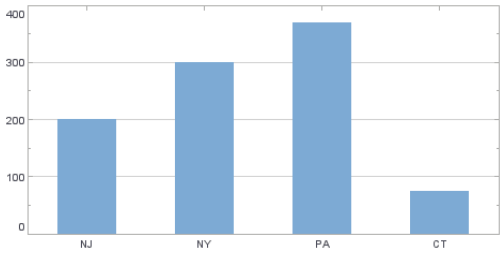
CT.14Representing Multiple Measures

To represent multiple measures on a single chart, simply define a *GraphElement* object for each measure, and add the *GraphElement* to the Chart.

Consider the following script.

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

```
dataset = [["State", "Quantity 1", "Quantity
2"],["NJ",200,500], ["NY",300,1000],["PA",370,440],
["CT",75,20]];
graph = new EGraph();
var elem1 = new IntervalElement("State", "Quantity 1");
graph.addElement(elem1);
```



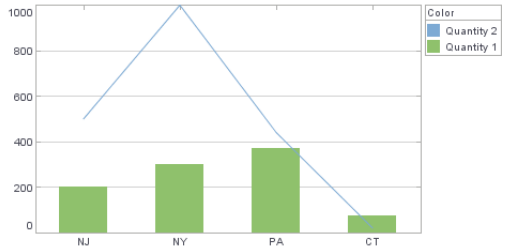
This creates a basic bar chart displaying the dimensions 'State' and 'Quantity 1'. To add the measure 'Quantity 2' to the chart as line element, follow the steps below:

1. Create a new *LineElement* to represent the graph of 'Quantity 2' vs. 'State'.

   ```
   var elem2 = new LineElement("State", "Quantity 2");
   ```

2. Assign this new element to the main Chart object.

   ```
   graph.addElement(elem2);
   ```



A legend is created automatically. To modify the legend, make changes to the *VisualFrame*'s *LegendSpec* property. See *Changing Legend Properties*.

See Also

Changing Legend Properties, for more details about modifying legends.
Representing Data with Shape, Color, Size, to visually break out data as sub-series.

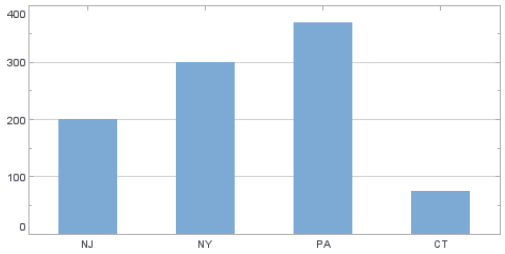
CT.15Adding Decorative Elements

To draw decorative elements (lines, shapes, text, etc.) on the chart, use a *GraphForm* object. Consider the following example:

```
Note: Script that modifies 'graph' should be placed at the element
level. See Adding Element-Level Script in Report Scripting and
Adding Component Script in Dashboard Scripting for more
information.

      dataset = [["State", "Quantity"],["NJ", 200],["NY", 300],
      ["PA", 370],["CT", 75]];
      graph = new EGraph();
```

```
var elem = new IntervalElement("State", "Quantity");
graph.addElement(elem);
```



This generates a basic bar chart with quantities for four different states. To add a small note indicating that the lowest value was due to an inventory problem, follow the steps below.

1. Create a new *LineForm* object, and specify location values to point at the 'CT' bar.

   ```
   var lineform = new LineForm();
   lineform.addValues([['CT', 150],['CT', 100]]);
   ```

2. Set the line color to red, and draw an arrow at the end.

   ```
   lineform.setColor(java.awt.Color(0xff0000));
   lineform.setEndArrow(true);
   ```

3. Create a new *LabelForm* object, and specify location values to position it above the 'CT' bar.

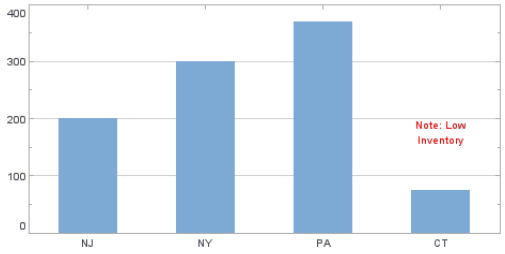
   ```
   var labelform = new LabelForm();
   labelform.setValues(['CT', 150]);
   ```

4. Set the label contents, set the text color to red, and center-align. To set the text color, create a new *TextSpec* object and assign it to the *LabelForm*.

   ```
   var labelSpec = new TextSpec();
   labelSpec.setColor(java.awt.Color(0xff0000));
   labelform.setTextSpec(labelSpec);
   labelform.setLabel("Note: Low\nInventory");
   labelform.setAlignmentX(Chart.CENTER_ALIGNMENT);
   ```

5. Assign the *LineForm* and *LabelForm* objects to the Chart object.

   ```
   graph.addForm(lineform);
   graph.addForm(labelform);
   ```



The final script is shown below.

```
dataset = [["State", "Quantity"],["NJ", 200],["NY",
300],["PA", 370],["CT", 75]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var lineform = new LineForm();
lineform.addValues([['CT', 150],['CT', 100]]);
lineform.setColor(java.awt.Color(0xff0000));
lineform.setEndArrow(true);
var labelform = new LabelForm();
labelform.setValues(['CT', 150]);

var labelSpec = new TextSpec();
labelSpec.setColor(java.awt.Color(0xff0000));
labelform.setTextSpec(labelSpec);
labelform.setLabel("Note: Low\nInventory");
labelform.setAlignmentX(Chart.CENTER_ALIGNMENT);
graph.addForm(lineform);
graph.addForm(labelform);
graph.addElement(elem);
```

Appendix CR: Chart Script Reference

This appendix lists all of the JavaScript objects and properties available for use in Style Chart scripts. Many of these properties are also exposed within Style Studio's **Report Element** panel.

CR.1 Chart Object

The Chart element presents datasets as visual graphs.

See Also

Modifying Chart Properties, for examples of setting chart properties in script.

Modifying a Chart Data Binding, for examples of modifying field bindings.

Modifying a Chart Element using API Functions, for examples of modifying the contents of a chart.

CR.1.1 General Properties

This section presents general Chart properties.

QUERY

Specifies the query bound to the Chart element.

Type

```
string
```

Example

```
query = 'customers';
```

See Also

Modifying a Chart Data Binding, for an example of data binding.

GRAPH

Specifies the *EGraph* object to be rendered by the Chart element.

Type

EGraph

Example

```
graph = new EGraph();
```

See Also

Creating a Chart Using API Functions, for an example of how to create a chart.

SETHYPERLINK(COL, HYPERLINK)

Adds hyperlinks to a chart dataset. Use `setParameterField` to specify values to pass as parameters in the link and `setTargetFrame` to specify the link's target frame. See the example below.

Parameter

col	dataset index, first dataset is "1"
hyperlink	string or inetsoft.report.Hyperlink.Ref object

Example 1

```
var link = new inetsoft.report.Hyperlink("www.google.com",
inetsoft.report.Hyperlink.WEB_LINK);
link.setParameterField("test", "Sales");
link.setTargetFrame("_blank");
setHyperlink(1, link);
```

Alternatively, you can pass all of this information within a single array as the second argument to the function. See the example below. The Boolean item in the array indicates whether the hyperlink targets a report/Viewsheet (`true`) or a web site (`false`).

Example 2

```
setHyperlink(1, ["report1", [["param1", "value1"],
["param2", "value2"],...]], true, "_blank")
```

CR.1.2 General Axis Properties

This section presents the general properties available for both *dimensions* and *measures*. Properties should be prefixed by one of the following:

```
axis['Dimension Name']
   or
axis['Measure Name']
```

Example

```
Chart1.axis['Company'].lineColor = [0,255,255];
Chart1.axis['Sum(Total)'].lineColor = [0,255,255];
```

AXIS.LABELVISIBLE

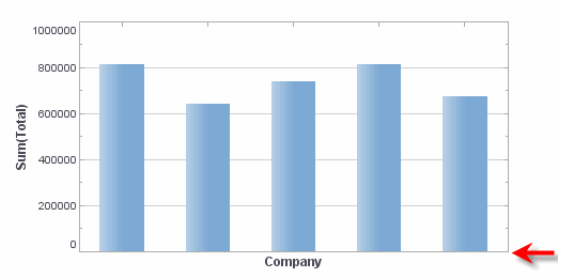
Sets the visibility of the dimension or measure axis labels.

Type

`Boolean`

Example

```
Chart1.axis['Company'].labelVisible = false;
```



AXIS.LINEVISIBLE

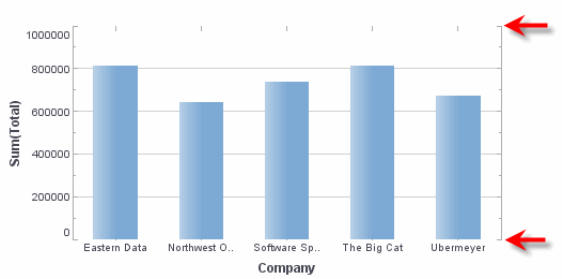
Sets the visibility of the dimension or measure axis line.

Type

`Boolean`

Example

```
Chart1.axis['Company'].lineVisible = false;
```



AXIS.TICKSVISIBLE

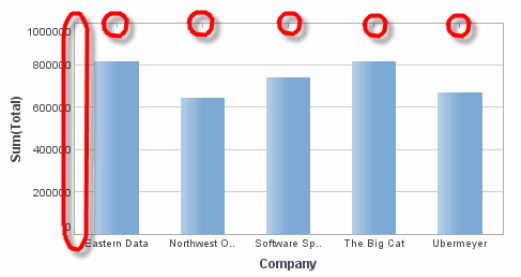
Sets the visibility of the axis tick marks for a dimension or measure.

Type

`Boolean`

Example

```
Chart1.axis['Sum(Total)'].ticksVisible = false
Chart1.axis['Company'].ticksVisible = true
```



AXIS.LINECOLOR

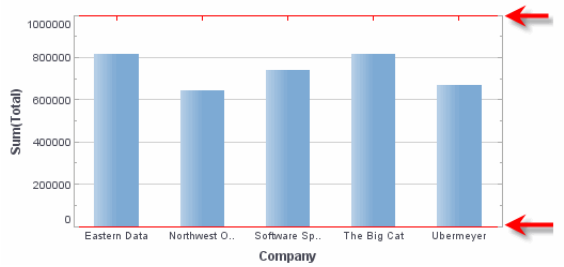
Sets the color of the dimension or measure axis line.

Type

```
java.awt.Color          e.g., java.awt.Color.BLUE
number (hex)            e.g., 0xFF0000
string (color name)     e.g., 'red'
array [r,g,b]           e.g., [255,0,0]
JSON {r:#,g:#,b:#}      e.g., {r:255,g:0,b:0}
```

Example

```
Chart1.axis['Company'].lineColor = [255,0,0];
Chart1.axis['Company'].lineColor = {r:255,g:0,b:0}
```



AXIS.LABELCOLOR

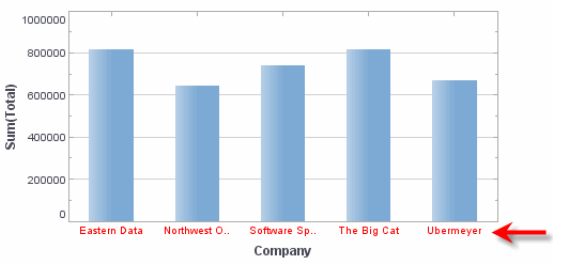
Sets the color of the dimension or measure axis labels.

Type

```
java.awt.Color          e.g., java.awt.Color.BLUE
number (hex)            e.g., 0xFF0000
string (color name)     e.g., 'red'
array [r,g,b]           e.g., [255,0,0]
JSON {r:#,g:#,b:#}      e.g., {r:255,g:0,b:0}
```

Example

```
Chart1.axis['Company'].labelColor = [255,0,0];
Chart1.axis['Company'].labelColor = {r:255,g:0,b:0}
```



AXIS.FONT

Use 'Edit Value' in the Script Editor's auto-complete menu to specify font descriptions.

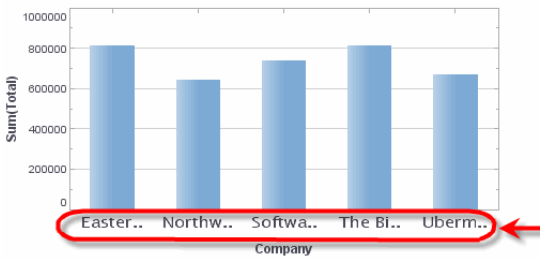
Sets the font of the dimension or measure axis labels.

Type

`java.awt.Font`, or
font description

Example

```
Chart1.axis['Company'].font =
   new java.awt.Font('Serif', java.awt.Font.BOLD, 12);
Chart1.axis['Company'].font ='Lucida Sans Unicode-BOLD-14'
```



AXIS.ROTATION

Use 'Edit Value' in the Script Editor's auto-complete menu to specify font descriptions.

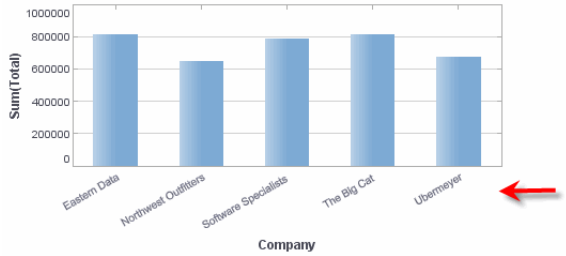
Sets the rotation of the dimension or measure axis labels.

Type

number (degrees)

Example

```
Chart1.axis['Company'].rotation = 30;
```



AXIS.FORMAT

Sets the format for dimension or measure axis labels. The available formats are shown below:

- `Chart.DATE_FORMAT` for date display
- `Chart.DECIMAL_FORMAT` for number display
- `Chart.MESSAGE_FORMAT` for text display
- `Chart.PERCENT_FORMAT` for fraction display
- `Chart.CURRENCY_FORMAT` for locale-adapting currency display

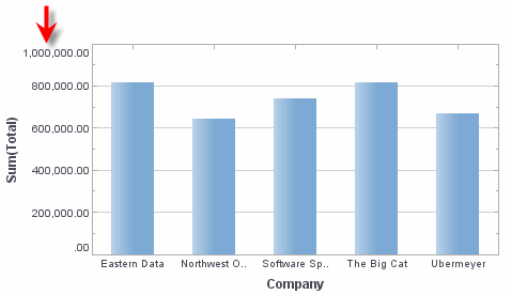
For the Date, Decimal, and Message formats, optionally specify the format and mask as items in an array. See *Format Types* in *Advanced Report Design* and *Format Tab* in *Advanced Dashboard Design* for more information about the masks.

Type

```
Constant (format) or Array ([format, mask])
```

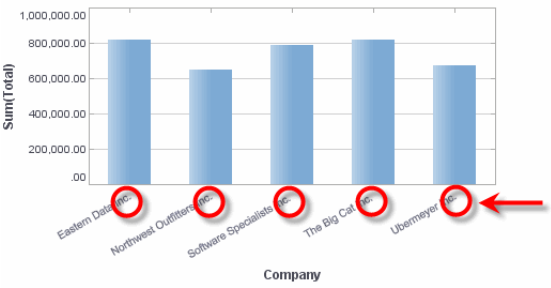
Example (Number Format)

```
Chart1.axis['Sum(Total)'].format =
[Chart.DECIMAL_FORMAT,"#,###.00"];
```



Example (Text Format)

```
Chart1.axis['Company'].format = [Chart.MESSAGE_FORMAT,"{0}
Inc."];
Chart1.axis['Company'].rotation = 30;
```



CR.1.3 Measure Axis Properties

This section presents the properties available specifically for a measure axis. Properties should be prefixed by `axis['Measure Name']`.

Example

```
Chart1.axis['Sum(Total)'].increment = 300000
```

AXIS.MAXIMUM

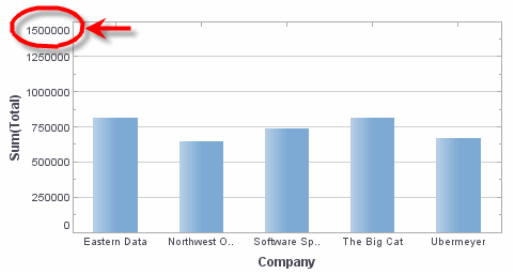
Sets the maximum value for a measure axis.

Type

```
number
```

Example

```
Chart1.axis['Sum(Total)'].maximum = 1500000;
```



AXIS.MINIMUM

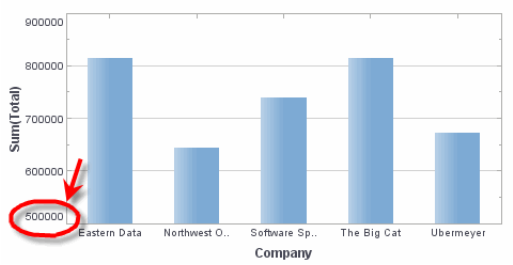
Sets the minimum value for a measure axis.

Type

```
number
```

Example

```
Chart1.axis['Sum(Total)'].minimum = 500000
```



AXIS.INCREMENT

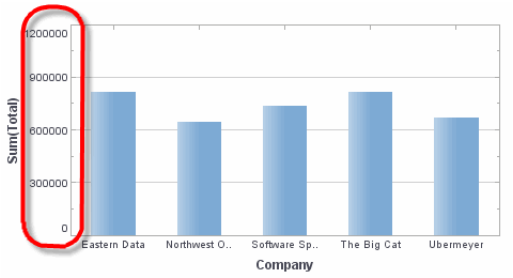
Sets the numerical increment for measure axis labels.

Type

```
number
```

Example

```
Chart1.axis['Sum(Total)'].increment = 300000
```



AXIS.MINORINCREMENT

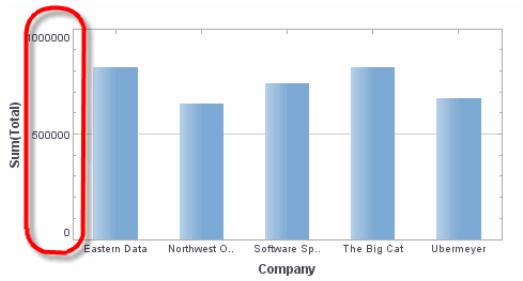
Sets the numerical increment for measure axis tick marks.

Type

`number`

Example

```
Chart1.axis['Sum(Total)'].increment = 500000
Chart1.axis['Sum(Total)'].minorIncrement = 100000
```



AXIS.LOGARITHMIC

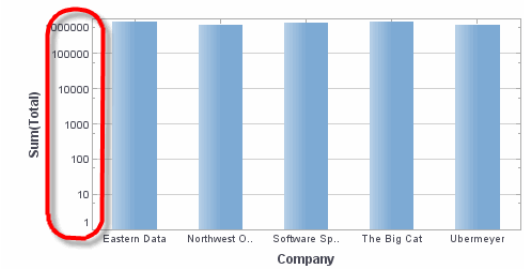
Sets logarithmic scaling for a measure axis.

Type

`Boolean`

Example

```
Chart1.axis['Sum(Total)'].logarithmic = true
```



AXIS.REVERSED

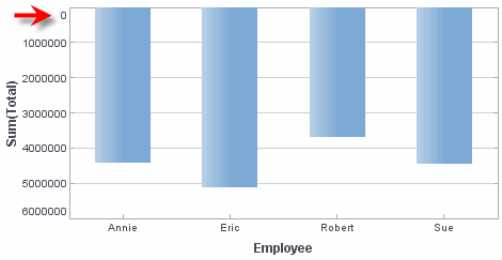
Reverses the direction of the axis scale so that higher numbers appear at the bottom or left.

Type

`Boolean`

Example

```
Chart1.axis['Sum(Total)'].reversed = true;
```



AXIS.SHAREDRANGE

For a facet chart (a chart with multiple bound X-fields and Y-fields), specifies whether the same scale range should be used across all sub-graphs in a facet, or whether a sub-graph's scaling should be shared only with sub-graphs in the same row (for Y-axis scaling) or same column (for X-axis scaling).

Type

`Boolean`

Example

```
Chart1.yAxis.sharedRange = false;
```



XAXIS

Provides a shortcut to set axis properties for the all X-axes.

Example

```
Chart1.xAxis.labelColor=[255,0,0];
Chart1.xAxis.rotation=30;
```

The available properties are listed in *General Axis Properties* and *Measure Axis Properties*.

To set properties for a particular X-axis (e.g., 'Category'), use the `axis` property together with the axis name:

```
Chart1.axis.Category.labelColor=[255,0,0];
```

```
Chart1.axis.Category.rotation=30;
```

See Also
yaxis, to set properties for Y-axes.

YAXIS

Provides a shortcut to set axis properties for all Y-axes.

Example

```
Chart1.yAxis.minorIncrement = 1000;
Chart1.yAxis.labelColor = [255,0,0]
```

The available properties are listed in *General Axis Properties* and *Measure Axis Properties*.

To set properties for a particular Y-axis (e.g., 'Total'), use the `axis` property together with the axis name:

```
Chart1.axis['Sum(Total)'].minorIncrement = 1000;
Chart1.axis['Sum(Total)'].labelColor = [255,0,0];
```

See Also
xaxis, to set properties for X-axes.
y2axis, to set properties for secondary Y-axes.

Y2AXIS

Provides a shortcut to set axis properties for all secondary Y-axes. This only applies to a measure axes that have been marked as 'Secondary Axis'.

Example

```
Chart1.y2Axis.minorIncrement=1000
Chart1.y2Axis.labelColor=[255,0,0]
```

The available properties are listed in *General Axis Properties* and *Measure Axis Properties*.

See Also
Single Chart vs. Separate Charts, in *Dashboard Design* or *Report Design* for information about setting a secondary axis.
yaxis, to set properties for primary Y-axes.

CR.1.4 Title Properties

This section presents properties available for Graph titles. Title properties should be prefixed by either `xTitle` or `yTitle`.

Example

```
Chart1.xTitle.text = "New Jersey Employees"
Chart1.yTitle.text = "New Jersey Sales"
```

XTITLE.TEXT

Sets the X-axis title text.

Type
```
string
```

Example
```
Chart1.xTitle.text = "New Jersey Employees"
```

YTITLE.TEXT

Sets the Y-axis title text.

Type
```
string
```

Example
```
Chart1.yTitle.text = "New Jersey Sales"
```

XTITLE.FONT

Use ‘Edit Value’ in the Script Editor’s auto-complete menu to specify font descriptions.

Sets the X-axis title font.

Type
```
java.awt.Font, or
font description
```

Example
```
Chart1.xTitle.font = new java.awt.Font('Serif',
java.awt.Font.BOLD, 12);
Chart1.xTitle.font = 'Lucida Sans Unicode-BOLD-14'
```

YTITLE.FONT

Use ‘Edit Value’ in the Script Editor’s auto-complete menu to specify font descriptions.

Sets the Y-axis title font.

Type
```
java.awt.Font, or
font description
```

Example
```
Chart1.yTitle.font = new java.awt.Font('Serif',
java.awt.Font.BOLD, 12);
Chart1.yTitle.font ='Lucida Sans Unicode-BOLD-14'
```

XTITLE.ROTATION

Sets the X-axis title rotation angle.

Type
```
number (degrees)
```

Example

```
Chart1.xTitle.rotation = 30;
```

YTITLE.ROTATION

Sets the Y-axis title rotation angle.

Type

```
number (degrees)
```

Example

```
Chart1.yTitle.rotation = 30;
```

XTITLE.COLOR

Sets the X-axis title color.

Type

```
java.awt.Color          e.g., java.awt.Color.BLUE
number (hex)            e.g., 0xFF0000
string (color name)     e.g., 'red'
array [r,g,b]           e.g., [255,0,0]
JSON {r:#,g:#,b:#}      e.g., {r:255,g:0,b:0}
```

Example

```
Chart1.xTitle.color = [128,0,128];
Chart1.xTitle.color = {r:128,g:0,b:128}
```

YTITLE.COLOR

Sets the Y-axis title color.

Type

```
java.awt.Color          e.g., java.awt.Color.BLUE
number (hex)            e.g., 0xFF0000
string (color name)     e.g., 'red'
array [r,g,b]           e.g., [255,0,0]
JSON {r:#,g:#,b:#}      e.g., {r:255,g:0,b:0}
```

Example

```
Chart1.yTitle.color = [128,0,128];
Chart1.yTitle.color = {r:128,g:0,b:128}
```

XTITLE.VISIBLE

Sets the visibility of the X-axis title.

Type

```
Boolean
```

Example

```
Chart1.xTitle.visible = false
Chart1.xTitle.visible = true
```

YTITLE.VISIBLE

Sets the visibility of the Y-axis title.

Type

```
Boolean
```

Example

```
Chart1.yTitle.visible = false
Chart1.yTitle.visible = true
```

X2TITLE.TEXT

Sets the title text for the secondary (top) X-axis.

Type

```
string
```

Example

```
Chart1.x2Title.text = "New Jersey Employees"
```

Y2TITLE.TEXT

Sets the title text for the secondary (right) Y-axis.

Type

```
string
```

Example

```
Chart1.y2Title.text = "New Jersey Sales"
```

X2TITLE.FONT

Use ‘Edit Value’ in the Script Editor’s auto-complete menu to specify font descriptions.

Sets the title font for the secondary (top) X-axis.

Type

```
java.awt.Font, or
font description
```

Example

```
Chart1.x2Title.font = new java.awt.Font('Serif',
java.awt.Font.BOLD, 12);
Chart1.x2Title.font ='Lucida Sans Unicode-BOLD-14'
```

Y2TITLE.FONT

Use ‘Edit Value’ in the Script Editor’s auto-complete menu to specify font descriptions.

Sets the title font for the secondary (right) Y-axis.

Type

```
java.awt.Font, or
font description
```

Example

```
Chart1.y2Title.font = new java.awt.Font('Serif',
java.awt.Font.BOLD, 12);
Chart1.y2Title.font ='Lucida Sans Unicode-BOLD-14'
```

X2TITLE.ROTATION

Sets the title rotation angle for the secondary (top) X-axis.

Type

```
number (degrees)
```

Example

```
Chart1.x2Title.rotation = 30;
```

Y2TITLE.ROTATION

Sets the title rotation angle for the secondary (right) Y-axis.

Type

```
number (degrees)
```

Example

```
Chart1.y2Title.rotation = 30;
```

X2TITLE.COLOR

Sets the title color for the secondary (top) X-axis.

Type

```
java.awt.Color          e.g., java.awt.Color.BLUE
number (hex)            e.g., 0xFF0000
string (color name)     e.g., 'red'
array [r,g,b]           e.g., [255,0,0]
JSON {r:#,g:#,b:#}      e.g., {r:255,g:0,b:0}
```

Example

```
Chart1.x2Title.color = [128,0,128];
Chart1.x2Title.color = {r:128,g:0,b:128}
```

Y2TITLE.COLOR

Sets the title color for the secondary (right) Y-axis.

Type

```
java.awt.Color          e.g., java.awt.Color.BLUE
number (hex)            e.g., 0xFF0000
string (color name)     e.g., 'red'
array [r,g,b]           e.g., [255,0,0]
JSON {r:#,g:#,b:#}      e.g., {r:255,g:0,b:0}
```

Example

```
Chart1.y2Title.color = [128,0,128];
Chart1.y2Title.color = {r:128,g:0,b:128}
```

X2TITLE.VISIBLE

Sets the visibility of the secondary (top) X-axis. The default is `true` (visible) if a value has been provided for the *x2Title.text*, otherwise `false`.

Type

`Boolean`

Example

```
Chart1.x2Title.visible = false
Chart1.x2Title.visible = true
```

Y2TITLE.VISIBLE

Sets the visibility of the secondary (right) Y-axis. The default is `true` (visible) if a value has been provided for the *y2Title.text*, otherwise `false`.

Type

`Boolean`

Example

```
Chart1.y2Title.visible = false
Chart1.y2Title.visible = true
```

CR.1.5 Plot Properties

This section presents properties available for the Graph *plot area*.

CHARTSTYLE

For a chart with multiple styles (‘Multiple Styles’ selected from ‘Select Chart Style’ panel), sets the style for the specified dataset. For a chart with a single style, sets the global style.

Type

Chart Constants constant
(see *Chart Styles*)

Example

```
Multi-style chart:
  Chart1.chartStyle['Sum(Total)'] = Chart.CHART_LINE
Single-style chart:
  Chart1.chartStyle = Chart.CHART_LINE
```

FILLTIMEGAP

When you designate a dimension as a “time series” (see *bindingInfo.setTimeSeries(col,boolean)*), the `fillTimeGap` property specifies how missing data is represented on the chart. When `fillTimeGap` is `false`, the plotted lines simply ignore the missing data and connect adjacent points on the chart. When `fillTimeGap` is `true`, the behavior for missing data is determined by the *fillZero* property.

Type

`Boolean`

The following examples display a time-series which is missing data for the month of April 2009. To create this time-series, bind a chart to the 'Sales' > 'Sales Explore' Worksheet. Place the 'Date' field on the X-axis (with 'Level' = 'Month') and place 'Total' on the Y-axis. To simulate missing data, add a condition to the chart to filter out data from April 2009. Use the scripts below to modify the way the chart represents the missing data for April 2009.

Example 1

```
Chart1.fillTimeGap = false
```



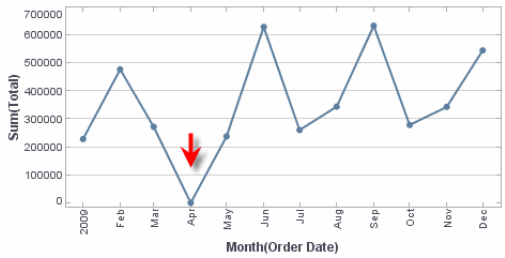
Example 2

```
Chart1.fillTimeGap = true
Chart1.fillZero = false
```



Example 3

```
Chart1.fillTimeGap = true
Chart1.fillZero = true
```



FILLZERO

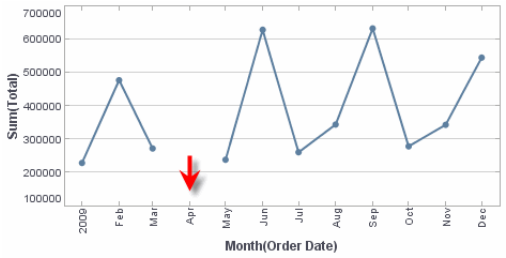
When you set *fillTimeGap* to `true` or set the corresponding property in the dialog box, the `fillZero` property determines whether gaps are represented as 'Null', which leaves an empty space at the location, or as 'Zero', which inserts a numerical value of 0 at the location.

Type

`Boolean`

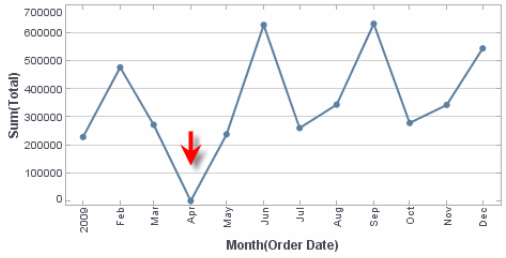
Example 1

```
Chart1.fillTimeGap = true
Chart1.fillZero = false
```



Example 2

```
Chart1.fillTimeGap = true
Chart1.fillZero = true
```



MAPTYPE

Specifies the type of map (region of globe displayed).

Type

```
Chart Constants constant or
String:       'U.S.', 'Asia', 'Canada', 'Europe', 'Mexico'
              'World'
```

Example

```
Chart1.mapType = Chart['MAP_TYPE_U.S.'];
Chart1.mapType = 'U.S.'
```

VALUEFONT

Use 'Edit Value' in the Script Editor's auto-complete menu to specify font descriptions.

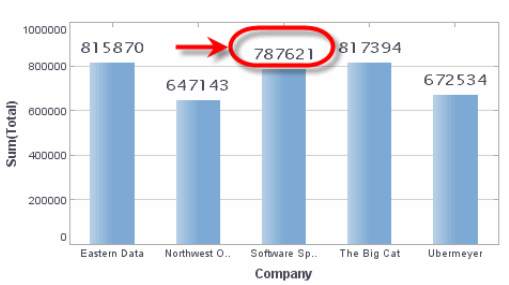
Sets the font for data values displayed in the chart plot area.

Type

`java.awt.Font`, or
font description

Example

```
Chart1.valueFont = new java.awt.Font('Serif',
java.awt.Font.BOLD, 12);
Chart1.valueFont='Lucida Sans Unicode-BOLD-14'
```



See Also

valueFormats, to set the font of a value on a multi-style chart.

VALUEFORMAT

Sets the format for data values displayed in the chart plot area. The available formats are shown below:

- `Chart.DATE_FORMAT` for date display
- `Chart.DECIMAL_FORMAT` for number display
- `Chart.MESSAGE_FORMAT` for text display
- `Chart.PERCENT_FORMAT` for fraction display
- `Chart.CURRENCY_FORMAT` for locale-adapting currency display

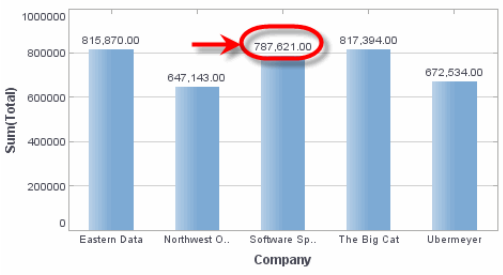
For the Date, Decimal, and Message formats, optionally specify the format and mask as items in an array. See *Format Types* in *Advanced Report Design* and *Format Tab* in *Advanced Dashboard Design* for more information about the masks.

Type

```
Constant (format) or Array ([format, mask])
```

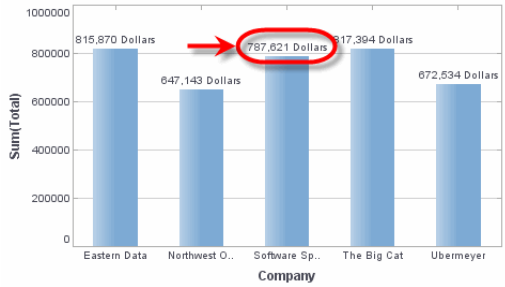
Example (Number Format)

```
Chart1.valueFormat = [Chart.DECIMAL_FORMAT,"#,###.00"];
```



Example (Text Format)

```
Chart1.valueFormat = [Chart.MESSAGE_FORMAT,"{0} Dollars"];
```



See Also

valueFormats, to set the format of a value on a multi-style chart.

VALUEROTATION

Sets the rotation angle for data values displayed in the chart plot area.

Type

```
number (degrees)
```

Example

```
Chart1.valueRotation = 30;
```



See Also

valueFormats, to set the rotation of a value on a multi-style chart.

VALUECOLOR

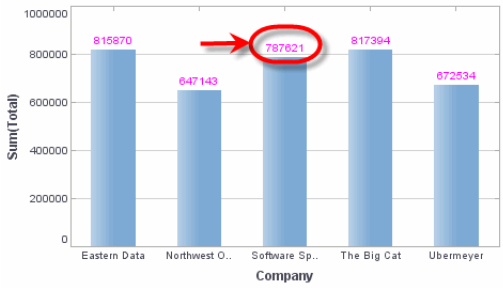
Sets the color for data values displayed in the chart plot area.

Type

```
java.awt.Color          e.g., java.awt.Color.BLUE
number (hex)            e.g., 0xFF0000
string (color name)     e.g., 'red'
array [r,g,b]           e.g., [255,0,0]
JSON {r:#,g:#,b:#}      e.g., {r:255,g:0,b:0}
```

Example

```
Chart1.valueColor = [255,0,255];
Chart1.valueColor = {r:255,g:0,b:255};
```



See Also

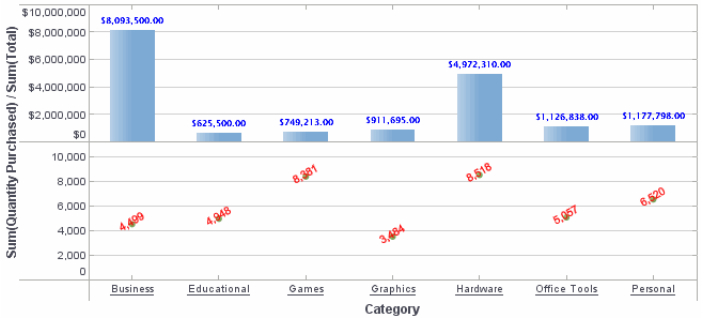
valueFormats, to set the color of a value on a multi-style chart.

VALUEFORMATS

Provides access to value formatting for Charts that have 'Multiple Styles' enabled. (See *Setting a Chart Style for an Individual Dataset* in *Advanced Dashboard Design* or *Setting a Chart Style for an Individual Dataset* in *Advanced Report Design* for information about the 'Multiple Styles' property.) To format a measure in a multiple-style chart, set the `valueFormats` property for the desired measure.

Example

```
Chart1.valueFormats['Sum(Quantity Purchased)'].color=
[255,0,0];
Chart1.valueFormats['Sum(Quantity Purchased)'].rotation =
30;
Chart1.valueFormats['Sum(Total)'].color = [0,0,255];
Chart1.valueFormats['Sum(Total)'].format =
[Chart.DECIMAL_FORMAT,"$#,###.00"]
Chart1.valueFormats['Sum(Total)'].font = 'Lucida Sans
Unicode-BOLD-8';
```



For information about the `valueFormats` sub-properties (`color`, `font`, `format`, `rotation`), see the following: *valueColor*, *valueFont*, *valueFormat*, *valueRotation*. To format *all* measures in a multiple-style chart, use one of these properties.

See Also

valueColor, to specify value color in a single-style chart.
valueFont, to specify value font in a single-style chart.
valueFormat, to specify value format in a single-style chart.
valueRotation, to specify value rotation in a single-style chart.

VALUEVISIBLE

Sets the visibility for data values displayed in the chart plot area.

Type

`Boolean`

Example

```
Chart1.valueVisible = false;
```

ADDCONFIDENCEINTERVALTARGET(MEASURE, COLORS, VALUES, OPTIONS)

Draws a set of horizontal or vertical confidence intervals on the chart.

Parameter

measure Name of measure on which to draw confidence intervals
colors Color (see *valueColor*) or array of colors to fill area of confidence intervals
values Confidence level as a percentage (e.g., '99' for 99% confidence interval)
options Name-value pairs of options (see below, e.g., {fillAbove:0xff0000, fillBelow:0x00ff00})

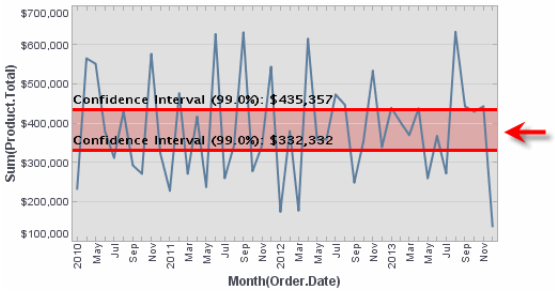
Options

fillAbove Color (see *valueColor*) to fill the region above the upper confidence line
fillBelow Color (see *valueColor*) to fill the region below the lower confidence line

label String to label the confidence line(s). The following codes can be inserted:
{0}: Target value
{1}: Target formula
{2}: Field name
lineColor Color (see *valueColor*) to apply to all confidence lines
lineStyle The line style (a *Line Styles* constant) to apply to all confidence lines
labelFont The font description (e.g., 'Lucida Sans Unicode-BOLD-14') to apply to all line labels
labelColor The color (see *valueColor*) to apply to all line labels

Example

```
var options =
{fillAbove:[225,225,225],fillBelow:[225,225,225],label:['{1
}: {0,number,$#,##0}','{1}:
{0,number,$#,##0}'],lineColor:'red',lineStyle:Chart.THICK_L
INE,labelFont:'Lucida Sans Unicode-BOLD-
11',labelColor:'black'}
addConfidenceIntervalTarget('Sum(Product.Total)',0xDDAAAA,
99,options)
```



See Also

Adding a Statistical Measure, in *Advanced Report Design*, for more information about target lines.
Adding a Statistical Measure, in *Advanced Dashboard Design*, for more information about target lines.

ADDPERCENTAGETARGET(MEASURE, COLORS, VALUES, OPTIONS)

Draws a set of horizontal or vertical lines on the chart to mark the specified percentage figures.

Parameter

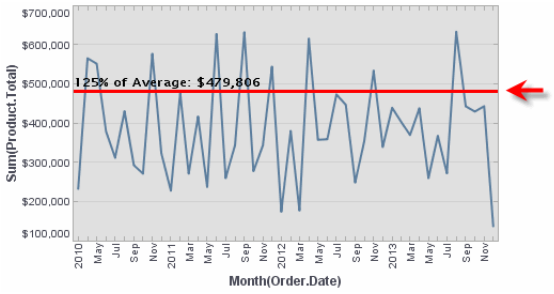
measure Name of measure on which to draw percentage lines
colors Color (see *valueColor*) or array of colors to fill areas between percentage lines
values Percentage, or array of percentages, at which to draw lines
options Name-value pairs of options (see below, e.g., {fillAbove:0xff0000, fillBelow:0x00ff00})

Options

fillAbove Color (see *valueColor*) to fill the region above the upper percentage line
fillBelow Color (see *valueColor*) to fill the region below the lower percentage line
label String, or array of strings, to label the percentage line(s). The following codes can be inserted:
{0}: Target value
{1}: Target formula
{2}: Field name
lineColor Color (see *valueColor*) to apply to all percentage lines
lineStyle The line style (a *Line Styles* constant) to apply to all percentage lines
labelFont The font description (e.g., 'Lucida Sans Unicode-BOLD-14') to apply to all line labels
labelColor The color (see *valueColor*) to apply to all line labels
percentageAggregate The basis on which to compute the percentage: 'Average', 'Minimum', 'Maximim', 'Median', 'Sum'

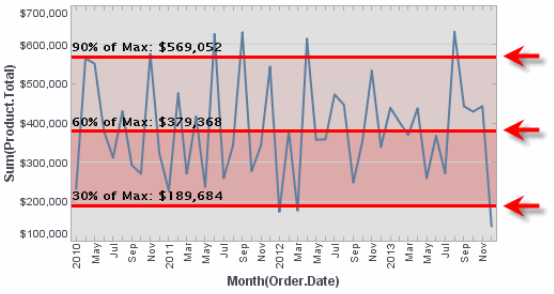
Example (Single Percentage Line)

```
var options =
{fillAbove:[225,225,225],fillBelow:[225,225,225],label:['{1
}:
{0,number,$#,##0}'],lineColor:'red',lineStyle:Chart.THICK_L
INE,labelFont:'Lucida Sans Unicode-BOLD-
11',labelColor:'black',percentageAggregate:'Average'}
addPercentageTarget('Sum(Product.Total)','null',125,option
s)
```



Example (Multiple Percentage Lines)

```
var options =
{fillAbove:[225,225,225],fillBelow:[225,225,225],label:['{1
}: {0,number,$#,##0}','{1}:
{0,number,$#,##0}'],lineColor:'red',lineStyle:Chart.THICK_L
INE,labelFont:'Lucida Sans Unicode-BOLD-
11',labelColor:'black',percentageAggregate:'Maximum'}
addPercentageTarget('Sum(Product.Total)',[0xDDAAAA,0xDDCCC
C],[30,60,90],options)
```



See Also

Adding a Statistical Measure, in *Advanced Report Design*, for more information about target lines.

Adding a Statistical Measure, in *Advanced Dashboard Design*, for more information about target lines.

ADDPERCENTILETARGET(MEASURE, COLORS, VALUES, OPTIONS)

Draws a set of horizontal or vertical lines on the chart to mark the specified percentiles.

Parameter

measure Name of measure on which to draw percentile lines
colors Color (see *valueColor*) or array of colors to fill areas between percentile lines
values Percentage, or array of percentages, at which to draw lines
options Name-value pairs of options (see below, e.g., {fillAbove:0xff0000, fillBelow:0x00ff00})

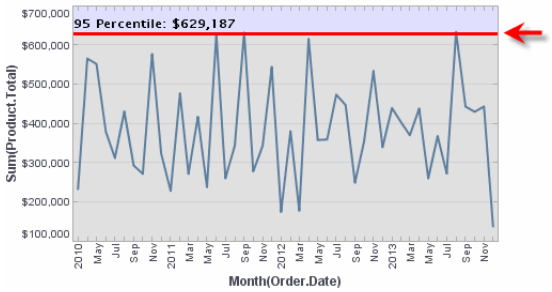
Options

fillAbove Color (see *valueColor*) to fill the region above the upper percentile line
fillBelow Color (see *valueColor*) to fill the region below the lower percentile line
label String, or array of strings, to label the percentile line(s). The following codes can be inserted:
{0}: Target value
{1}: Target formula
{2}: Field name
lineColor Color (see *valueColor*) to apply to all percentile lines
lineStyle The line style (a *Line Styles* constant) to apply to all percentile lines
labelFont The font description (e.g., 'Lucida Sans Unicode-BOLD-14') to apply to all line labels
labelColor The color (see *valueColor*) to apply to all line labels

Example (Single Percentile Line)

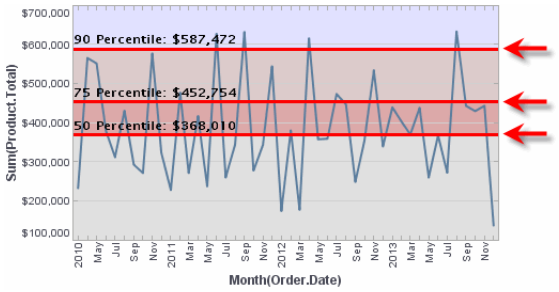
```
var options =
{fillAbove:[225,225,255],fillBelow:[225,225,225],label:['{1
}:
{0,number,$#,##0}'],lineColor:'red',lineStyle:Chart.THICK_L
```

```
INE,labelFont:'Lucida Sans Unicode-BOLD-
11',labelColor:'black'}
addPercentileTarget('Sum(Product.Total)','null',95,options
)
```



Example (Multiple Percentage Lines)

```
var options =
{fillAbove:[225,225,255],fillBelow:[225,225,225],label:['{1
}: {0,number,$#,##0}','{1}: {0,number,$#,##0}','{1}:
{0,number,$#,##0}'],lineColor:'red',lineStyle:Chart.THICK_L
INE,labelFont:'Lucida Sans Unicode-BOLD-
11',labelColor:'black'}
addPercentileTarget('Sum(Product.Total)',[0xDDAAAA,0xDDCCC
C],[50,75,90],options)
```



See Also

Adding a Statistical Measure, in *Advanced Report Design*, for more information about target lines.

Adding a Statistical Measure, in *Advanced Dashboard Design*, for more information about target lines.

ADDQUANTILETARGET(MEASURE, COLORS, VALUES, OPTIONS)

Draws a set of horizontal or vertical lines on the chart to mark the specified quantiles.

Parameter

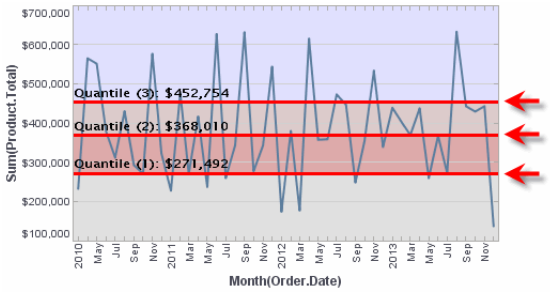
measure Name of measure on which to draw quantile lines
colors Color (see *valueColor*) or array of colors to fill areas between quantile lines
values Number of quantiles at which to draw lines
options Name-value pairs of options (see below, e.g., {fillAbove:0xff0000, fillBelow:0x00ff00})

Options

fillAbove Color (see *valueColor*) to fill the region above the upper quantile line
fillBelow Color (see *valueColor*) to fill the region below the lower quantile line
label String, or array of strings, to label the quantile line(s). The following codes can be inserted:
{0}: Target value
{1}: Target formula
{2}: Field name
lineColor Color (see *valueColor*) to apply to all quantile lines
lineStyle The line style (a *Line Styles* constant) to apply to all quantile lines
labelFont The font description (e.g., 'Lucida Sans Unicode-BOLD-14') to apply to all line labels
labelColor The color (see *valueColor*) to apply to all line labels

Example

```
var options =
{fillAbove:[225,225,255],fillBelow:[225,225,225],label:['{1
}: {0,number,$#,##0}','{1}: {0,number,$#,##0}','{1}:
{0,number,$#,##0}'],lineColor:'red',lineStyle:Chart.THICK_L
INE,labelFont:'Lucida Sans Unicode-BOLD-
11',labelColor:'black'}
addQuantileTarget('Sum(Product.Total)',[0xDDAAAA,0xDDCCCC]
,4,options)
```



See Also

Adding a Statistical Measure, in *Advanced Report Design*, for more information about target lines.
Adding a Statistical Measure, in *Advanced Dashboard Design*, for more information about target lines.

ADDSTANDARDDEVIATIONTARGET(MEASURE, COLORS, VALUES, OPTIONS)

Draws a set of horizontal or vertical lines on the chart to mark the specified number of standard deviations from the mean.

Parameter

measure Name of measure on which to draw standard deviation lines
colors Color (see *valueColor*) or array of colors to fill areas between standard deviation lines

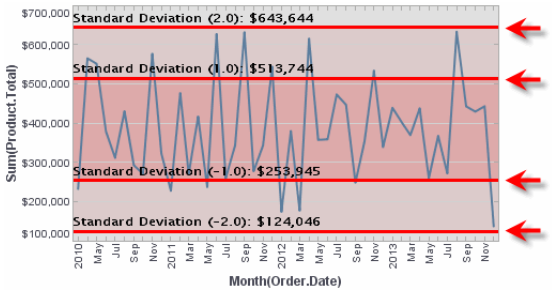
values Array of multipliers for which to draw standard deviation lines
options Name-value pairs of options (see below, e.g., {fillAbove:0xff0000, fillBelow:0x00ff00})

Options

fillAbove Color (see *valueColor*) to fill the region above the upper standard deviation line
fillBelow Color (see *valueColor*) to fill the region below the lower standard deviation line
label String, or array of strings, to label the standard deviation line(s). The following codes can be inserted:
{0}: Target value
{1}: Target formula
{2}: Field name
lineColor Color (see *valueColor*) to apply to all standard deviation lines
lineStyle The line style (a *Line Styles* constant) to apply to all standard deviation lines
labelFont The font description (e.g., 'Lucida Sans Unicode-BOLD-14') to apply to all line labels
labelColor The color (see *valueColor*) to apply to all line labels

Example

```
var options =
{fillAbove:[225,225,225],fillBelow:[225,225,225],label:['{1
}: {0,number,$#,##0}','{1}: {0,number,$#,##0}','{1}:
{0,number,$#,##0}','{1}:
{0,number,$#,##0}'],lineColor:'red',lineStyle:Chart.THICK_L
INE,labelFont:'Lucida Sans Unicode-BOLD-
11',labelColor:'black'}
addStandardDeviationTarget('Sum(Product.Total)',[0xDDCCCC,
0xDDAAAA,0xDDCCCC],[-1,1,-2,2],options)
```



See Also

Adding a Statistical Measure, in *Advanced Report Design*, for more information about target lines.
Adding a Statistical Measure, in *Advanced Dashboard Design*, for more information about target lines.

ADDTARGETBAND(MEASURE, COLORS, VALUES, OPTIONS)

Draws a set of horizontal or vertical target bands in the plot area using the specified colors and options.

Parameter

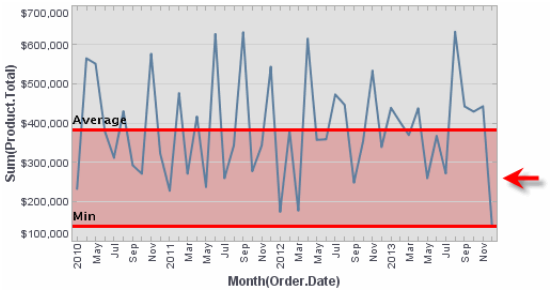
measure Name of measure on which to draw target bands
colors Color (see *valueColor*) or array of colors to fill area of bands
values Array of values and/or valid aggregation methods ('max','min','avg','med') at which to draw the band lines
options Name-value pairs of options (see below, e.g., {fillAbove:0xff0000, fillBelow:0x00ff00})

Options

fillAbove Color (see *valueColor*) to fill the region above the upper band line
fillBelow Color (see *valueColor*) to fill the region below the lower band line
label String to label the target band(s). The following codes can be inserted:
{0}: Target value
{1}: Target formula
{2}: Field name
lineColor Color (see *valueColor*) to apply to all target band lines
lineStyle The line style (a *Line Styles* constant) to apply to all target band lines
labelFont The font description (e.g., 'Lucida Sans Unicode-BOLD-14') to apply to all band labels
labelColor The color (see *valueColor*) to apply to all band labels

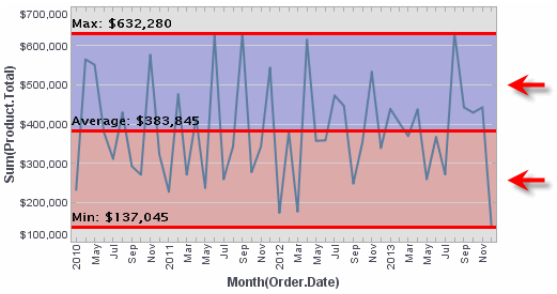
Example (One Target Band)

```
var options =
{fillAbove:[225,225,225],fillBelow:[225,225,225],label:['{1
}','{1}'],lineColor:'red',lineStyle:Chart.THICK_LINE,labelF
ont:'Lucida Sans Unicode-BOLD-11',labelColor:'black'}
addTargetBand('Sum(Product.Total)',0xDDAAAA,['min','avg'],
options)
```



Example (Multiple Target Bands)

```
var options =
{fillAbove:[225,225,225],fillBelow:[225,225,225],label:['{1
}: {0,number,$#,##0}','{1}: {0,number,$#,##0}','{1}:
{0,number,$#,##0}'],lineColor:'red',lineStyle:Chart.THICK_L
INE,labelFont:'Lucida Sans Unicode-BOLD-
11',labelColor:'black'}
addTargetBand('Sum(Product.Total)',[0xDDAAAA,0xAAAADD],['m
in','avg','max'],options)
```



See Also

Adding a Target Band, in *Advanced Report Design*, for more information about target lines.

Adding a Target Band, in *Advanced Dashboard Design*, for more information about target lines.

ADDTARGETLINE(MEASURE, COLORS, VALUES, OPTIONS)

Draws a set of horizontal or vertical target lines in the plot area using the specified colors and options.

Parameter

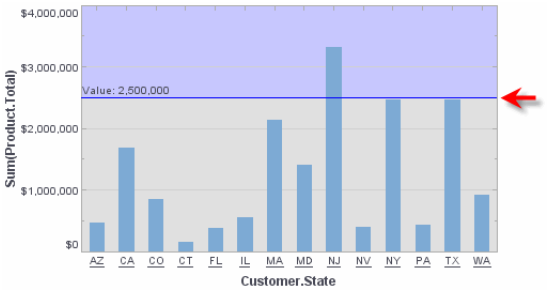
measure Name of measure on which to draw target line
colors Color (see *valueColor*) or array of colors to fill area between target lines
values Array of values and/or valid aggregation methods ('max','min','avg','med') at which to draw the lines
options Name-value pairs of options (see below, e.g., {fillAbove:0xff0000, fillBelow:0x00ff00})

Options

fillAbove Color (see *valueColor*) to fill the region above the target line
fillBelow Color (see *valueColor*) to fill the region below the target line
label String to label the target line(s). The following codes can be inserted:
{0}: Target value
{1}: Target formula
{2}: Field name
lineColor Color (see *valueColor*) to apply to all target lines
lineStyle The line style (a *Line Styles* constant) to apply to all target lines
labelFont The font description (e.g., 'Lucida Sans Unicode-BOLD-14') to apply to all target labels
labelColor The color (see *valueColor*) to apply to all target labels

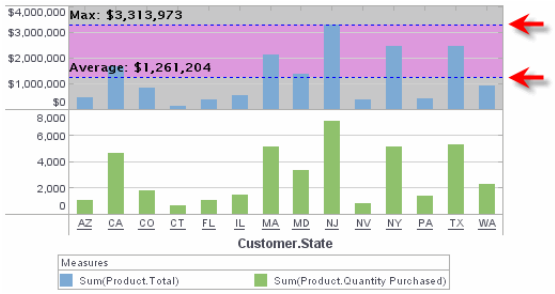
Example (One Target Line)

```
var options =
{fillAbove:[200,200,255],fillBelow:[225,225,225],label:'Val
ue: {0}',lineColor:'blue'}
addTargetLine('Sum(Product.Total)',null,2500000,options)
```



Example (Multiple Target Lines)

```
var options =
{fillAbove:[200,200,200],fillBelow:[200,200,200],label:['{1
}: {0,number,$#,##0}','{1}:
{0,number,$#,##0}'],lineColor:'blue',lineStyle:Chart.DASH_L
INE,labelFont:'Lucida Sans Unicode-BOLD-
11',labelColor:'black'}
addTargetLine('Sum(Product.Total)',0xDD99DD,['avg','max'],
options)
```



See Also

Adding a Target Line, in *Advanced Report Design*, for more information about target lines.

Adding a Target Line, in *Advanced Dashboard Design*, for more information about target lines.

APPLYEFFECT

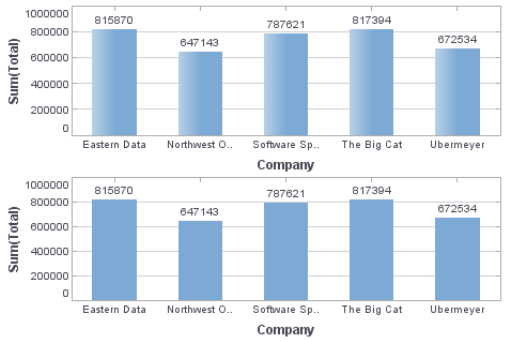
Enables a 3d visual effect for chart elements.

Type

`Boolean`

Example

```
Chart1.applyEffect = true
Chart2.applyEffect = false
```



PLOTALPHA

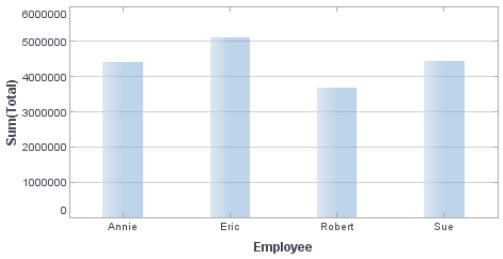
Sets the transparency of chart elements (bars, lines, points, etc.). A value of 0 indicates complete transparency (i.e., chart elements not visible), and a value of 1 indicates complete opacity.

Type

`Number` `Number in range [0,1]`

Example

```
Chart1.plotAlpha = .5;
```



PIEEXPLODED

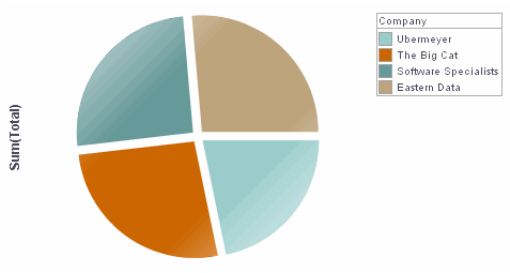
Separates the slices of a pie chart by a small gap.

Type

`Boolean`

Example

```
Chart1.pieExploded = true
```



XGRIDCOLOR

Sets the X-axis grid color.

Type

```
java.awt.Color          e.g., java.awt.Color.BLUE
number (hex)            e.g., 0xFF0000
string (color name)     e.g., 'red'
array [r,g,b]           e.g., [255,0,0]
JSON {r:#,g:#,b:#}      e.g., {r:255,g:0,b:0}
```

Example

```
Chart1.xGridColor = [255,0,0];
Chart1.xGridColor = {r:255,g:0,b:0}
```

YGRIDCOLOR

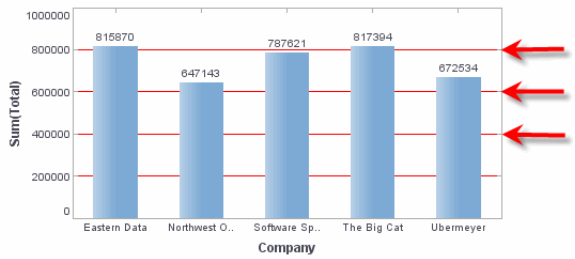
Sets the Y-axis grid color.

Type

```
java.awt.Color          e.g., java.awt.Color.BLUE
number (hex)            e.g., 0xFF0000
string (color name)     e.g., 'red'
array [r,g,b]           e.g., [255,0,0]
JSON {r:#,g:#,b:#}      e.g., {r:255,g:0,b:0}
```

Example

```
Chart1.yGridColor = [255,0,0];
Chart1.yGridColor = {r:255,g:0,b:0}
```



XGRIDSTYLE

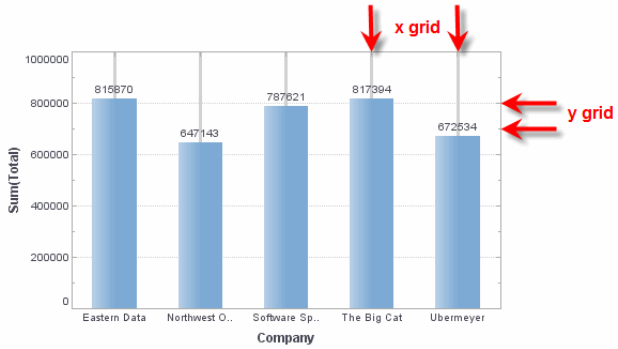
Sets the X-axis grid style. A grid line is drawn at each axis tick.

Type

Chart Constants constant
(see *Line Styles*)

Example

```
Chart1.yGridStyle=Chart.DOT_LINE
Chart1.xGridStyle=Chart.THICK_LINE
```



YGRIDSTYLE

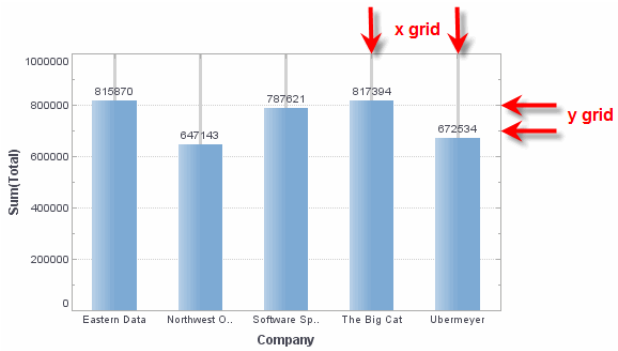
Sets the Y-axis grid style. A grid line is drawn at each axis tick.

Type

Chart Constants constant
(see *Line Styles*)

Example

```
Chart1.yGridStyle=Chart.DOT_LINE
Chart1.xGridStyle=Chart.THICK_LINE
```



XBANDCOLOR

Displays bands of the specified color on a categorical or timescale X-axis.

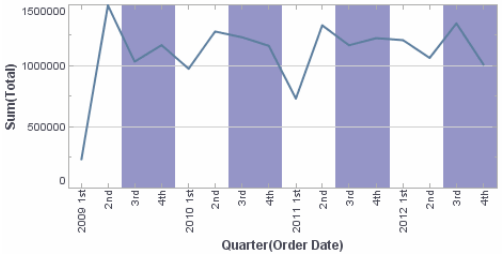
Type

```
java.awt.Color          e.g., java.awt.Color.BLUE
number (hex)            e.g., 0xFF0000
string (color name)     e.g., 'red'
array [r,g,b]           e.g., [255,0,0]
```

```
JSON {r:#,g:#,b:#}      e.g., {r:255,g:0,b:0}
```

Example

```
Chart1.xBandColor = [150,150,200];
Chart1.xBandColor = {r:150,g:150,b:200};
```



See Also

xBandSize, to set the band thickness.

YBANDCOLOR

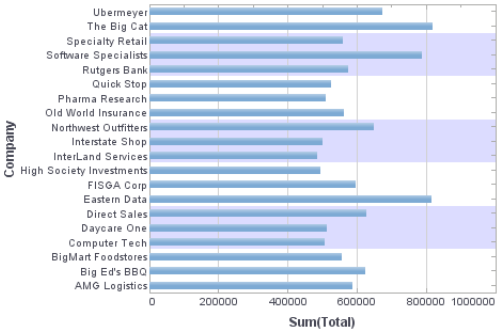
Displays bands of the specified color on a categorical or timescale Y-axis.

Type

```
java.awt.Color          e.g., java.awt.Color.BLUE
number (hex)            e.g., 0xFF0000
string (color name)     e.g., 'red'
array [r,g,b]           e.g., [255,0,0]
JSON {r:#,g:#,b:#}      e.g., {r:255,g:0,b:0}
```

Example

```
Chart1.yBandColor = [220,220,255];
Chart1.yBandColor = {r:220,g:220,b:255};
```



See Also

yBandSize, to set the band thickness.

XBANDSIZE

Specifies the width of the bands on a categorical or timescale X-axis. The band size is an integer indicating the number of category groups (for a

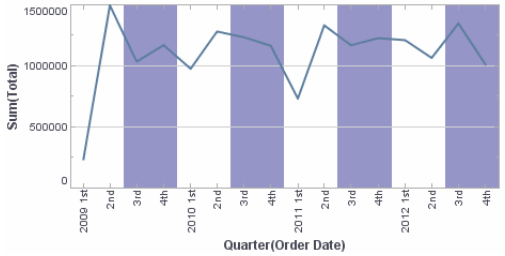
categorical axis) or the number of date groups (for a timescale axis) to be spanned by each band.

Type

```
integer                number of groups spanned by band
```

Example

```
Chart1.xBandSize = 2;
```



See Also

xBandColor, to set the band color.

YBANDSIZE

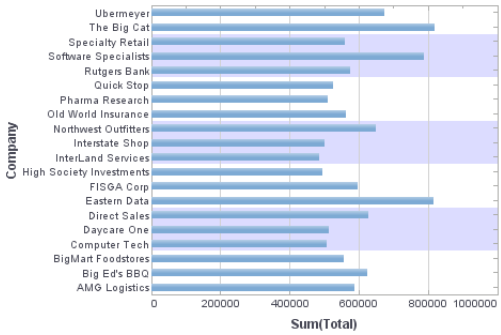
Specifies the width of the bands on a categorical or timescale Y-axis. The band size is an integer indicating the number of category groups (for a categorical axis) or the number of date groups (for a timescale axis) to be spanned by each band.

Type

```
integer                number of groups spanned by band
```

Example

```
Chart1.yBandSize = 3;
```



See Also

yBandColor, to set the band color.

QUADRANTCOLOR

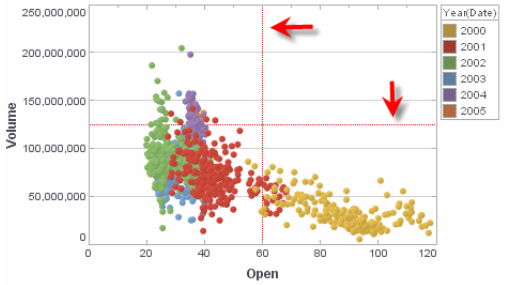
Sets the color of the quadrant lines.

Type

```
java.awt.Color          e.g., java.awt.Color.BLUE
number (hex)            e.g., 0xFF0000
string (color name)     e.g., 'red'
array [r,g,b]           e.g., [255,0,0]
JSON {r:#,g:#,b:#}      e.g., {r:255,g:0,b:0}
```

Example

```
Chart1.quadrantColor=[255,0,0];
Chart1.quadrantStyle=Chart.DOT_LINE;
```



QUADRANTSTYLE

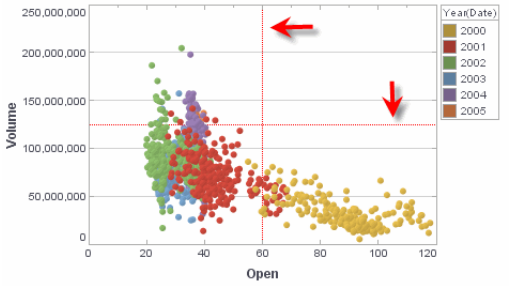
Sets the style of the quadrant lines.

Type

Chart Constants constant
(see *Line Styles*)

Example

```
Chart1.quadrantColor=[255,0,0];
Chart1.quadrantStyle=Chart.DOT_LINE;
```



DIAGONALCOLOR

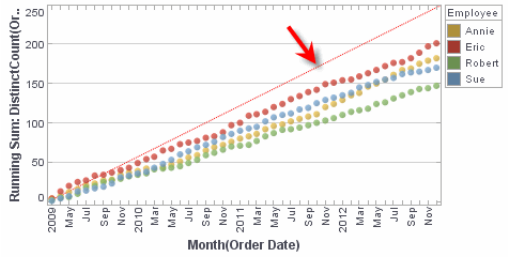
Sets the color of the diagonal line.

Type

```
java.awt.Color          e.g., java.awt.Color.BLUE
number (hex)            e.g., 0xFF0000
string (color name)     e.g., 'red'
array [r,g,b]           e.g., [255,0,0]
JSON {r:#,g:#,b:#}      e.g., {r:255,g:0,b:0}
```

Example

```
Chart1.diagonalColor=[255,0,0];
Chart1.diagonalStyle=Chart.DOT_LINE;
```



DIAGONALSTYLE

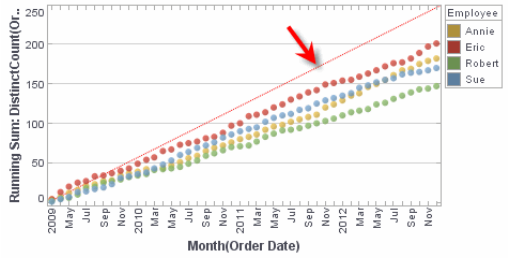
Sets the style of the diagonal line.

Type

Chart Constants constant
(see *Line Styles*)

Example

```
Chart1.diagonalColor=[255,0,0];
Chart1.diagonalStyle=Chart.DOT_LINE;
```



LEGENDPOSITION

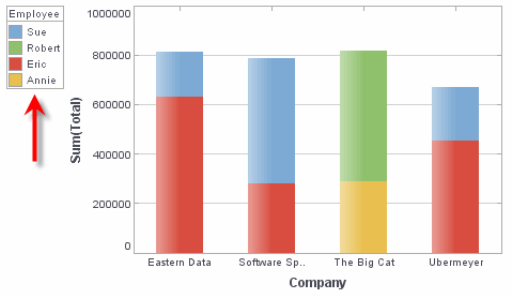
Sets the legend position relative to the chart. Set the property to `Chart.NONE` to hide the legend.

Type

Chart Constants constant
(see *Legend Positions*)

Example

```
Chart1.legendPosition = Chart.LEFT;
```



LEGENDBORDER

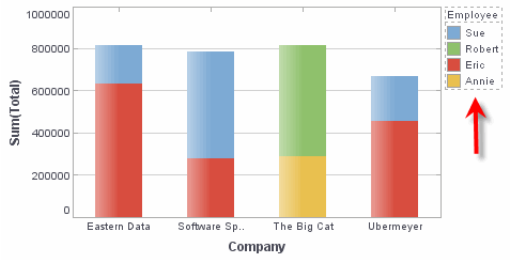
Sets the legend border line style. Set the property to `Chart.NONE` to hide the legend border.

Type

Chart Constants constant
(see *Line Styles*)

Example

```
Chart1.legendBorder = Chart.DASH_LINE;
```



LEGENDBORDERCOLOR

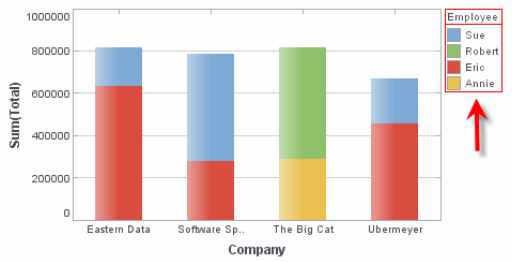
Sets the color of the legend border.

Type

```
java.awt.Color          e.g., java.awt.Color.BLUE
number (hex)            e.g., 0xFF0000
string (color name)     e.g., 'red'
array [r,g,b]           e.g., [255,0,0]
JSON {r:#,g:#,b:#}      e.g., {r:255,g:0,b:0}
```

Example

```
Chart1.legendBorderColor = [255,0,0];
Chart1.legendBorderColor = {r:255,g:0,b:0}
```



LEGENDTITLEFONT

Use 'Edit Value' in the Script Editor's auto-complete menu to specify font descriptions.

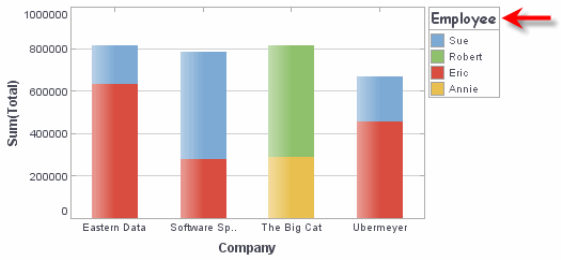
Sets the font for the legend title.

Type

`java.awt.Font`, or
font description

Example

```
Chart1.legendTitleFont = new java.awt.Font('Serif',
java.awt.Font.BOLD, 12);
Chart1.legendTitleFont = 'Comic Sans MS-BOLD-14';
```



LEGENDTITLECOLOR

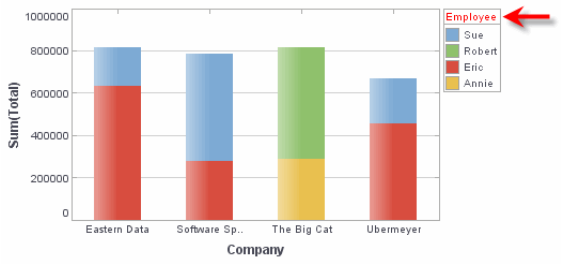
Sets the color of the legend title text.

Type

`java.awt.Color`	`e.g., java.awt.Color.BLUE`
`number (hex)`	`e.g., 0xFF0000`
`string (color name)`	`e.g., 'red'`
`array [r,g,b]`	`e.g., [255,0,0]`
`JSON {r:#,g:#,b:#}`	`e.g., {r:255,g:0,b:0}`

Example

```
Chart1.legendTitleColor = [255,0,0];
Chart1.legendTitleColor = {r:255,g:0,b:0}
```



COLORLEGEND.FONT

Use 'Edit Value' in the Script Editor's auto-complete menu to specify font descriptions.

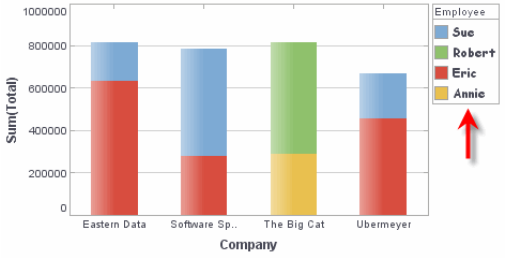
Sets the font for labels within a color-key legend.

Type

`java.awt.Font`, or
font description

Example

```
Chart1.colorLegend.font = new java.awt.Font('Serif',
java.awt.Font.BOLD, 12);
Chart1.colorLegend.font = 'Comic Sans MS-BOLD-12';
```



SHAPELEGEND.FONT

Use 'Edit Value' in the Script Editor's auto-complete menu to specify font descriptions.

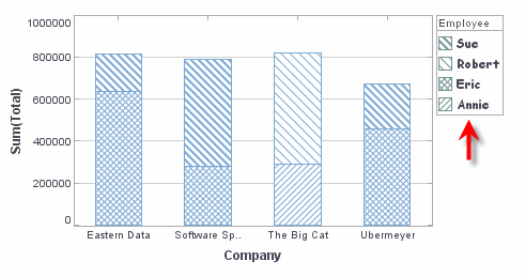
Sets the font for labels within a shape-key legend.

Type

`java.awt.Font`, or
font description

Example

```
Chart1.shapeLegend.font = new java.awt.Font('Serif',
java.awt.Font.BOLD, 12);
Chart1.shapeLegend.font = 'Comic Sans MS-BOLD-12';
```



SIZELEGEND.FONT

Use 'Edit Value' in the Script Editor's auto-complete menu to specify font descriptions.

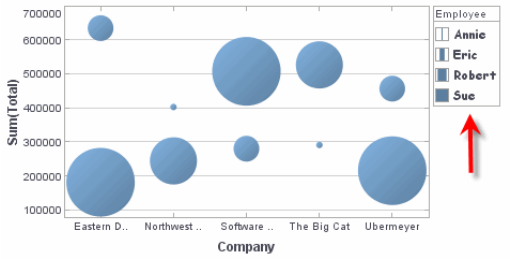
Sets the font for labels within a size-key legend.

Type

`java.awt.Font`, or
font description

Example

```
Chart1.sizeLegend.font = new java.awt.Font('Serif',
java.awt.Font.BOLD, 12);
Chart1.sizeLegend.font = 'Comic Sans MS-BOLD-12';
```



COLORLEGEND.COLOR

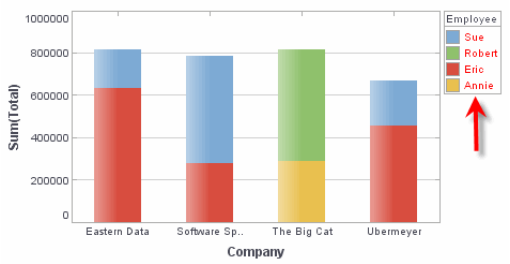
Sets the color for labels within a color-key legend.

Type

`java.awt.Color`	e.g., `java.awt.Color.BLUE`
number (hex)	e.g., `0xFF0000`
string (color name)	e.g., `'red'`
array [r,g,b]	e.g., `[255,0,0]`
JSON {r:#,g:#,b:#}	e.g., `{r:255,g:0,b:0}`

Example

```
Chart1.colorLegend.color = [255,0,0];
Chart1.colorLegend.color = {r:255,g:0,b:0}
```



SHAPELEGEND.COLOR

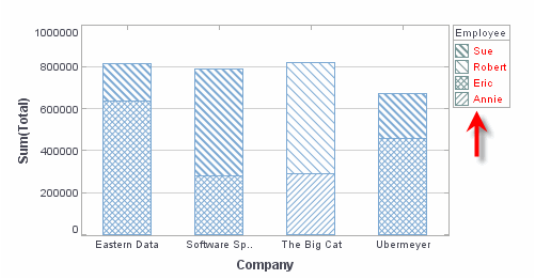
Sets the color for labels within a shape-key legend.

Type

```
java.awt.Color          e.g., java.awt.Color.BLUE
number (hex)            e.g., 0xFF0000
string (color name)     e.g., 'red'
array [r,g,b]           e.g., [255,0,0]
JSON {r:#,g:#,b:#}      e.g., {r:255,g:0,b:0}
```

Example

```
Chart1.shapeLegend.color = [255,0,0];
Chart1.shapeLegend.color = {r:255,g:0,b:0}
```



SIZELEGEND.COLOR

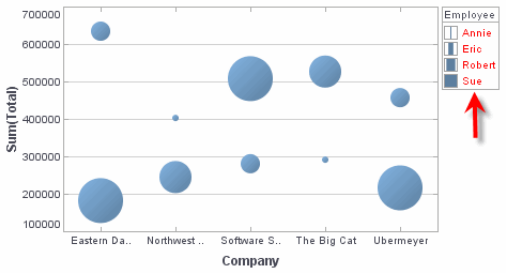
Sets the color for labels within a size-key legend.

Type

```
java.awt.Color          e.g., java.awt.Color.BLUE
number (hex)            e.g., 0xFF0000
string (color name)     e.g., 'red'
array [r,g,b]           e.g., [255,0,0]
JSON {r:#,g:#,b:#}      e.g., {r:255,g:0,b:0}
```

Example

```
Chart1.sizeLegend.color = [255,0,0];
Chart1.sizeLegend.color = {r:255,g:0,b:0}
```



COLORLEGEND.TITLE

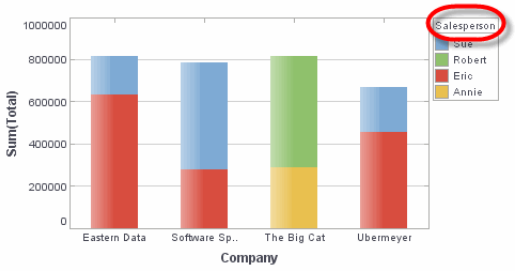
Sets the legend title for a color-key legend.

Type

`String`

Example

```
Chart1.colorLegend.title = 'Salesperson';
```



SHAPELEGEND.TITLE

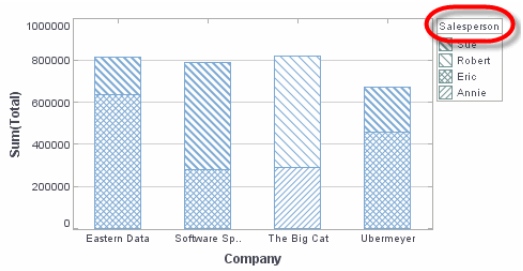
Sets the legend title for a shape-key legend.

Type

`String`

Example

```
Chart1.shapeLegend.title = 'Salesperson';
```



SIZELEGEND.TITLE

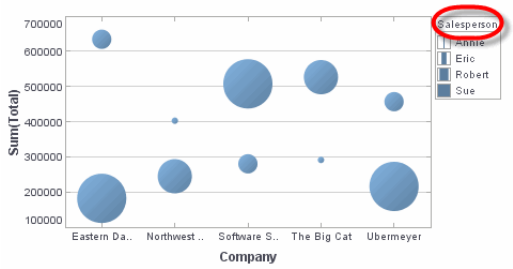
Sets the legend title for a size-key legend.

Type

`String`

Example

```
Chart1.sizeLegend.title = 'Salesperson';
```



COLORLEGEND.FORMAT

Sets the format for data values displayed in the color legend. The available formats are shown below:

- `Chart.DATE_FORMAT` for date display
- `Chart.DECIMAL_FORMAT` for number display
- `Chart.MESSAGE_FORMAT` for text display
- `Chart.PERCENT_FORMAT` for fraction display
- `Chart.CURRENCY_FORMAT` for locale-adapting currency display

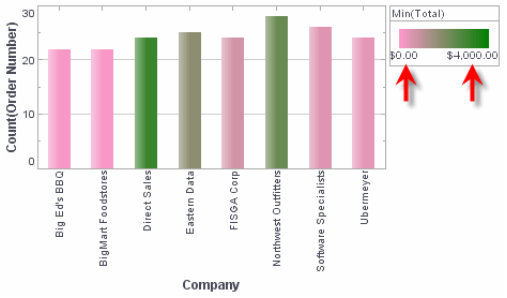
For the Date, Decimal, and Message formats, optionally specify the format and mask as items in an array. See *Format Types* in *Advanced Report Design* and *Format Tab* in *Advanced Dashboard Design* for more information about the masks.

Type

`Constant (format) or Array ([format, mask])`

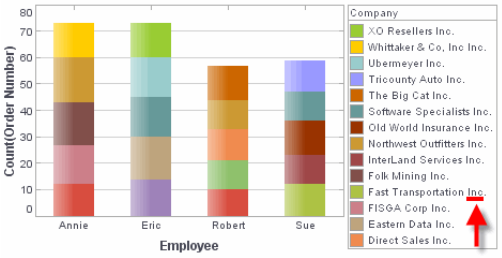
Example (Number Format)

```
Chart1.colorLegend.format =
[Chart.DECIMAL_FORMAT,"$#,##0.00"]
```



Example (Text Format)

```
Chart1.colorLegend.format = [Chart.MESSAGE_FORMAT,"{0}
Inc."];
```



SHAPELEGEND.FORMAT

Sets the format for data values displayed in the shape legend. The available formats are shown below:

- `Chart.DATE_FORMAT` for date display
- `Chart.DECIMAL_FORMAT` for number display
- `Chart.MESSAGE_FORMAT` for text display
- `Chart.PERCENT_FORMAT` for fraction display
- `Chart.CURRENCY_FORMAT` for locale-adapting currency display

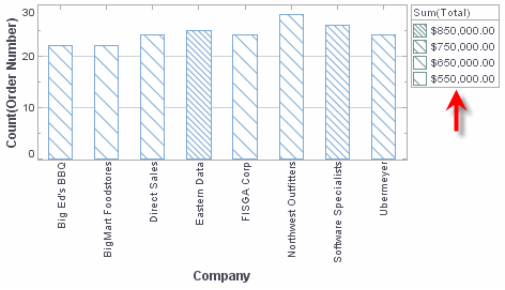
For the Date, Decimal, and Message formats, optionally specify the format and mask as items in an array. See *Format Types* in *Advanced Report Design* and *Format Tab* in *Advanced Dashboard Design* for more information about the masks.

Type

```
Constant (format) or Array ([format, mask])
```

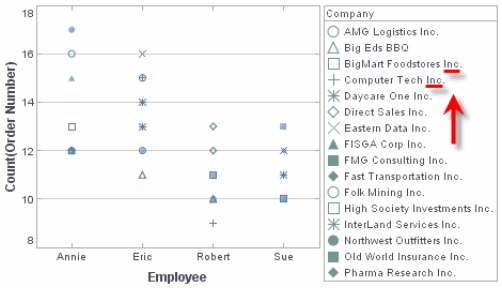
Example (Number Format)

```
Chart1.shapeLegend.format =
[Chart.DECIMAL_FORMAT,"$#,##0.00"];
```



Example (Text Format)

```
Chart1.shapeLegend.format = [Chart.MESSAGE_FORMAT,"{0}
Inc."];
```



SIZELEGEND.FORMAT

Sets the format for data values displayed in the size legend. The available formats are shown below:

- `Chart.DATE_FORMAT` for date display
- `Chart.DECIMAL_FORMAT` for number display
- `Chart.MESSAGE_FORMAT` for text display
- `Chart.PERCENT_FORMAT` for fraction display
- `Chart.CURRENCY_FORMAT` for locale-adapting currency display

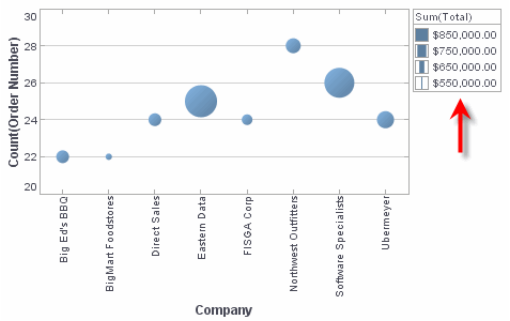
For the Date, Decimal, and Message formats, optionally specify the format and mask as items in an array. See *Format Types* in *Advanced Report Design* and *Format Tab* in *Advanced Dashboard Design* for more information about the masks.

Type

```
Constant (format) or Array ([format, mask])
```

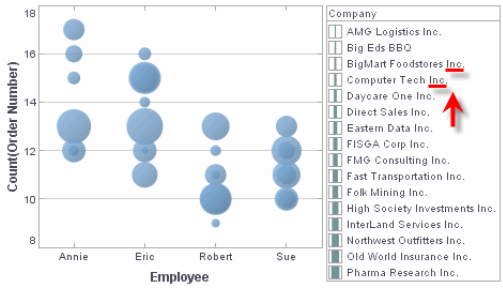
Example (Number Format)

```
Chart1.sizeLegend.format =
[Chart.DECIMAL_FORMAT,"$#,##0.00"];
```



Example (Text Format)

```
Chart1.sizeLegend.format = [Chart.MESSAGE_FORMAT,"{0}
Inc."];
```



COLORFIELD

Returns the name of the query field that is currently assigned to the ‘Color’ binding (*ColorFrame*) at runtime. This property is read-only.

Type

```
String
```

Example

```
if (Chart1.colorField == 'State') {
  StateSelectionList1.visible = true;
}
else {
  StateSelectionList1.visible = false;
}
```

See Also

bindingInfo.setColorField(col, type), to set the ‘Color’ binding in script.
bindingInfo.getColorField(), to return the design-time ‘Color’ binding.

SHAPEFIELD

Returns the name of the query field that is currently assigned to the ‘Shape’ binding (*ShapeFrame*) at runtime. This property is read-only.

Type

```
String
```

Example

```
if (Chart1.shapeField == 'State') {
  StateSelectionList1.visible = true;
}
else {
  StateSelectionList1.visible = false;
}
```

See Also

bindingInfo.setShapeField(col, type), to set the 'Shape' binding in script.
bindingInfo.getShapeField(), to return the design-time 'Shape' binding.

SIZEFIELD

Returns the name of the query field that is currently assigned to the 'Size' binding (*SizeFrame*) at runtime. This property is read-only.

Type

```
String
```

Example

```
if (Chart1.sizeField == 'State') {
  StateSelectionList1.visible = true;
}
else {
  StateSelectionList1.visible = false;
}
```

See Also

bindingInfo.setSizeField(col, type), to set the 'Size' binding in script.
bindingInfo.getSizeField(), to return the design-time 'Size' binding.

TEXTFIELD

Returns the name of the query field that is currently assigned to the 'Text' binding (*TextFrame*) at runtime. This property is read-only.

Type

```
String
```

Example

```
if (Chart1.textField == 'State') {
  StateSelectionList1.visible = true;
}
else {
  StateSelectionList1.visible = false;
}
```

See Also

bindingInfo.setTextField(col, type), to set the 'Text' binding in script.
bindingInfo.getTextField(), to return the design-time 'Text' binding.

GEOFIELDS

Returns the names of the query fields (as an array) that are currently assigned to the 'Geographic' binding at runtime. This property is read-only.

Type

```
Array of Strings
```

Example

```
if (Chart1.geoFields[0] == 'State') {
  StateSelectionList1.visible = true;
}
else {
  StateSelectionList1.visible = false;
}
```

See Also

bindingInfo.geoFields, to associate fields with the 'Geographic' binding.

XFIELDS

Returns the names of the query fields (as an array) that are currently assigned to the X-axis binding at runtime. This property is read-only.

Type

```
Array of Strings
```

Example

```
if (Chart1.xFields[0] == 'State') {
  StateSelectionList1.visible = true;
}
else {
  StateSelectionList1.visible = false;
}
```

See Also

bindingInfo.xFields, to associate fields with the X-axis binding.

YFIELDS

Returns the names of the query fields (as an array) that are currently assigned to the Y-axis binding at runtime. This property is read-only.

Type

```
Array of Strings
```

Example

```
if (Chart1.yFields[0] == 'Quantity Purchased') {
  Gauge1.visible = true;
}
else {
  Gauge1.visible = false;
}
```

See Also

bindingInfo.yFields, to associate fields with the X-axis binding.

SETLABELALIASOFCOLORLEGEND(VALUE, ALIAS)

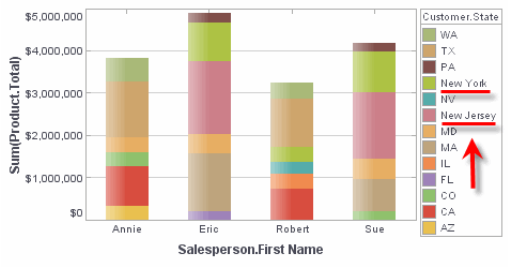
Replace the specified label in the color legend with an alias.

Parameters

value	Value to be replaced
alias	Replacement string

Example

```
Chart1.setLabelAliasOfColorLegend('NJ','New Jersey');
Chart1.setLabelAliasOfColorLegend('NY','New York');
```



SETLABELALIASOFSHAPELEGEND(VALUE, ALIAS)

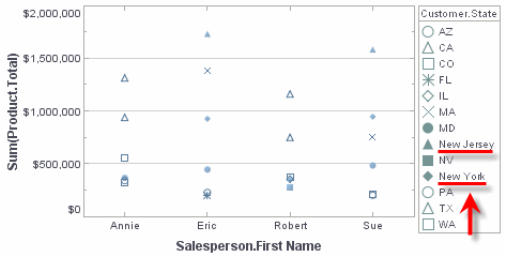
Replace the specified label in the shape legend with an alias.

Parameters

value	Value to be replaced
alias	Replacement string

Example

```
Chart1.setLabelAliasOfShapeLegend('NJ','New Jersey');
Chart1.setLabelAliasOfShapeLegend('NY','New York');
```



SETLABELALIASOFSIZELEGEND(VALUE, ALIAS)

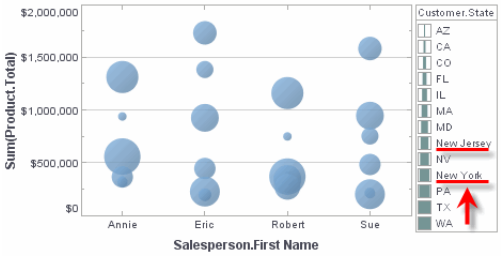
Replace the specified label in the size legend with an alias.

Parameters

value	Value to be replaced
alias	Replacement string

Example

```
Chart1.setLabelAliasOfSizeLegend('NJ','New Jersey');
Chart1.setLabelAliasOfSizeLegend('NY','New York');
```

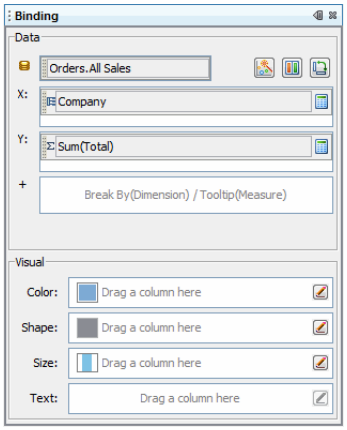


CR.1.6 The Chart bindingInfo Property

`Note: Report script that modifies 'bindingInfo' should be placed in the onLoad handler.`

This section presents functions related to the `bindingInfo` property of the Graph. The `bindingInfo` property provides access to the binding, grouping, and aggregation settings available within the Chart **Binding** panel.

Figure 1. The Binding panel.



The `bindingInfo` properties also control the *ColorFrame*, *ShapeFrame*, *SizeFrame*, and *TextureFrame* objects that determine visual styling for data represented on the chart (e.g., subseries).

BINDINGINFO.XFIELDS

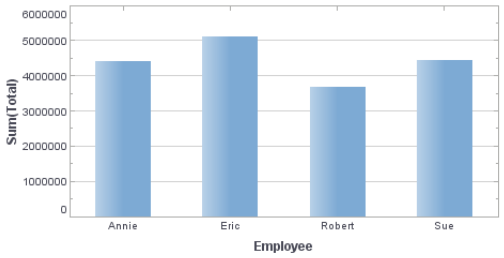
Specifies the query fields to be displayed on the X-axis.

Type

2D Array `[[Xfield1,Type1],[Xfield2,Type2],...]`
`Xfield: String`
`Type: Chart Constants constant: STRING, DATE, or NUMBER`

Example

```
Chart1.query = "All Sales";
Chart1.bindingInfo.xFields = [["Employee",Chart.STRING]];
Chart1.bindingInfo.yFields = [["Total",Chart.NUMBER]];
```



See Also

xFields, to return the fields currently bound to X-axis at runtime.

BINDINGINFO.YFIELDS

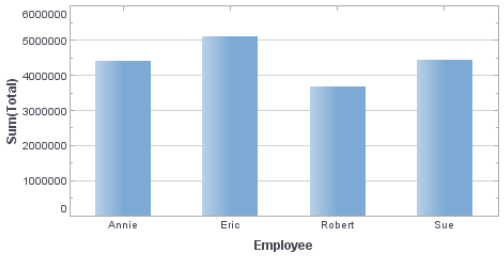
Specifies the query fields to be displayed on the Y-axis.

Type

2D Array `[[Xfield1,Type1],[Xfield2,Type2],...]`
`Xfield: String`
`Type: Chart Constants constant: STRING, DATE, or NUMBER`

Example

```
Chart1.query = "All Sales";
Chart1.bindingInfo.xFields = [["Employee",Chart.STRING]];
Chart1.bindingInfo.yFields = [["Total",Chart.NUMBER]];
```



See Also

yFields, to return the fields currently bound to X-axis at runtime.

BINDINGINFO.BREAKDOWNFIELDS

Specifies the query fields on which to break down the data. These fields will be bound to the 'Group By' region of the Chart Editor.

Type

2D Array `[[Field1,Type1],[Field2,Type2],...]`
`Field: String`
`Type: Chart Constants constant:`

STRING, DATE, or NUMBER

Example

```
Chart1.query = "All Sales";
Chart1.bindingInfo.xFields = [["Employee",Chart.STRING]];
Chart1.bindingInfo.yFields = [["Total",Chart.NUMBER]];
Chart1.bindingInfo.breakdownFields = [["Order
Date",Chart.DATE]];
```



BINDINGINFO.GEOFIELDS

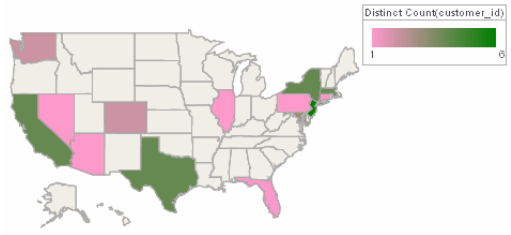
Specifies the query fields to be used as geographic fields. These fields will be bound to the 'Geographic' region of the Chart Editor.

Type

2D Array `[[Field1,Type1],[Field2,Type2],...]`
Field: String
Type: *Chart Constants* constant:
STRING, DATE, or NUMBER

Example

```
Chart1.query = "customers";
Chart1.chartStyle = Chart.CHART_MAP;
Chart1.bindingInfo.setColorField("customer_id",Chart.NUMBER
);
Chart1.bindingInfo.setFormula("customer_id",Chart.DISTINCTC
OUNT_FORMULA,Chart.AESTHETIC_COLOR);
Chart1.bindingInfo.geoFields = [['state',Chart.STRING]];
Chart1.mapType = Chart['MAP_TYPE_U.S.'];
Chart1.bindingInfo.setMapLayer("state",Chart.STATE);
Chart1.bindingInfo.shapes['DistinctCount(customer_id)'] =
new StaticShapeFrame(GShape.NIL);
```



See Also

geoFields, to return the currently bound 'Geographic' field at runtime.

BINDINGINFO.SETMAPLAYER(COL,LAYER)

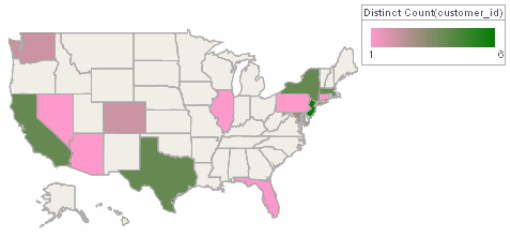
Specifies the map layer corresponding to a geographic field.

Parameters

col	Field for which layer is assigned
layer	*Chart Constants* constant: Chart.STATE Chart.CITY Chart.COUNTRY Chart.ZIP Chart.PROVINCE

Example

```
Chart1.query = "customers";
Chart1.chartStyle = Chart.CHART_MAP;
Chart1.bindingInfo.setColorField("customer_id",Chart.NUMBER
);
Chart1.bindingInfo.setFormula("customer_id",Chart.DISTINCTC
OUNT_FORMULA,Chart.AESTHETIC_COLOR);
Chart1.bindingInfo.geoFields = [['state',Chart.STRING]];
Chart1.mapType = Chart['MAP_TYPE_U.S.'];
Chart1.bindingInfo.setMapLayer("state",Chart.STATE);
Chart1.bindingInfo.shapes['DistinctCount(customer_id)']=new
StaticShapeFrame(GShape.NIL);
```



See Also

bindingInfo.getMapLayer(col), to return the layer assigned to a geographic field.

mapType, to set the outline of the map.

BINDINGINFO.GETMAPLAYER(COL)

Returns the map layer assigned to a geographic field.

Parameters

col	Field for which layer is desired

Example

```
var layer = Chart1.bindingInfo.getMapLayer('state')
```

See Also

bindingInfo.setMapLayer(col,layer), to set the geographic layer.

BINDINGINFO.GETGROUPORDER(COL,BINDING)

Returns the grouping level for a specified Date dimension.

Parameters

col Name of field for which to obtain grouping
binding The field's binding (*Chart Constants* constant):

```
Chart.BINDING_FIELD: bound to X or Y axis
Chart.AESTHETIC_COLOR: bound to Color
Chart.AESTHETIC_SHAPE: bound to Shape
Chart.AESTHETIC_SIZE: bound to Size
Chart.AESTHETIC_TEXT: bound to Text
```

Return

```
Number:    5 = Chart.YEAR_INTERVAL
           4 = Chart.QUARTER_INTERVAL
           3 = Chart.MONTH_INTERVAL
           2 = Chart.WEEK_INTERVAL
           1 = Chart.DAY_INTERVAL
           8 = Chart.HOUR_INTERVAL
           7 = Chart.MINUTE_INTERVAL
           6 = Chart.SECOND_INTERVA
         513 = Chart.QUARTER_OF_YEAR_PART
         514 = Chart.MONTH_OF_YEAR_PART
         515 = Chart.WEEK_OF_YEAR_PART
         518 = Chart.DAY_OF_MONTH_PART
         519 = Chart.DAY_OF_WEEK_PART
         521 = Chart.HOUR_OF_DAY_PART
```

Example

```
Chart1.bindingInfo.getGroupOrder("Order Date",
   Chart.BINDING_FIELD)
```

BINDINGINFO.SETGROUPORDER(COL,GROUPING)

Specifies the grouping level for a specified Date dimension.

Parameter

col date field to group
grouping *Chart Constants* constant:

```
Chart.YEAR_INTERVAL
Chart.QUARTER_INTERVAL
Chart.MONTH_INTERVAL
Chart.WEEK_INTERVAL
Chart.DAY_INTERVAL
Chart.HOUR_INTERVAL
Chart.MINUTE_INTERVAL
Chart.SECOND_INTERVAL
Chart.QUARTER_OF_YEAR_PART
Chart.MONTH_OF_YEAR_PART
Chart.WEEK_OF_YEAR_PART
Chart.DAY_OF_MONTH_PART
Chart.DAY_OF_WEEK_PART
Chart.HOUR_OF_DAY_PART
```

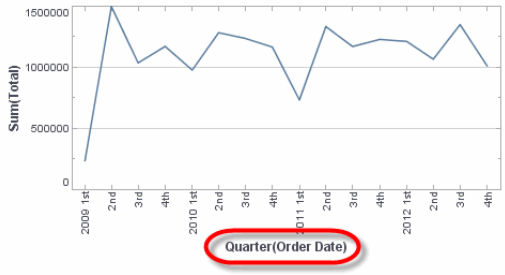
The distinction between grouping levels such as `Chart.MONTH_INTERVAL` and `Chart.MONTH_OF_YEAR_PART` is as follows:

- `Chart.MONTH_INTERVAL` groups the same month of different years individually (e.g., January 2009, January 2010, February 2009, February 2010, etc.).
- `Chart.MONTH_OF_YEAR_PART` groups the same month of different years together (January, February, etc.).

The same distinction obtains for `Chart.DAY_INTERVAL`, `Chart.DAY_OF_MONTH_PART`, and other similar pairs.

Example

```
Chart1.bindingInfo.setGroupOrder("Order
Date",Chart.QUARTER_INTERVAL)
```



BINDINGINFO.ISTIMESERIES(COL)

Returns the state of the specified Date column, designated as time-series or not.

Parameter

col	Name of field to test for time-series

Return

Boolean	true: Field is set as time-series false: Field is not set as time-series

Example

```
var ts = Chart1.bindingInfo.isTimeSeries("Order Date");
```

BINDINGINFO.SETTIMESERIES(COL,BOOLEAN)

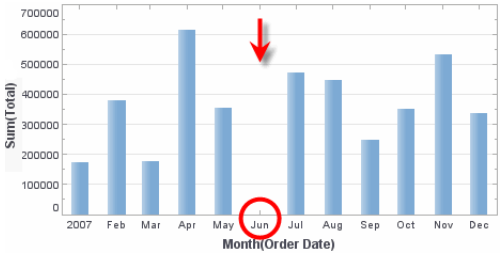
Specifies that gaps in Date data should be retained. For example, if the data in the chart is grouped by month, and there is no data for the month of June, the 'timeSeries' property ensures that the month of June is still retained in the chart.

Parameter

col	Name of field to set as time-series
Boolean	true: gaps are preserved for date data false: gaps are removed

Example

```
Chart1.bindingInfo.setTimeSeries("Order Date", true);
```



BINDINGINFO.GETCOLUMNORDER(COL)

Returns the sorting order for a specified dimension.

Parameter

col	Name of field for which to obtain sorting

Return

Number	1=ascending, 2=descending 0=none

Example

```
var ord = Chart1.bindingInfo.getColumnOrder("Company")
```

See Also

bindingInfo.getColumnOrder(colName), for the equivalent Table function.

BINDINGINFO.SETCOLUMNORDER(COL,ORDER)

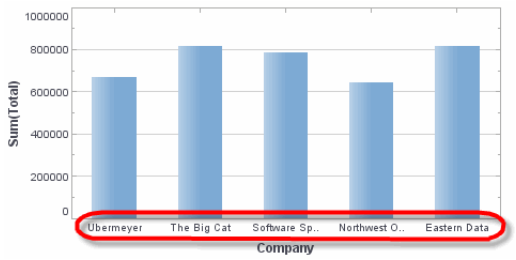
Specifies the sorting order for a specified dimension.

Parameter

col	Name of field for which to obtain sorting
order	*Chart Constants* constant: Chart.SORT_ASC // ascending Chart.SORT_DESC // descending Chart.SORT_NONE

Example

```
Chart1.bindingInfo.setColumnOrder("Company",
   Chart.SORT_DESC)
```



See Also

bindingInfo.setColumnOrder(colName,order), for the equivalent Table function.

BINDINGINFO.GETTOPN(COL)

Returns the number (N) of top or bottom groups being filtered for the specified field.

Parameter

col	Name of filtered field

Example

```
var N = Chart1.bindingInfo.getTopN("Company");
```

See Also

bindingInfo.getTopN(groupCol), for the equivalent Table function.

BINDINGINFO.SETTOPN(COL,N)

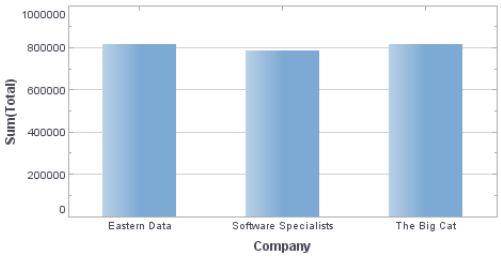
Filters the top or bottom N groups for a specified field based on an aggregate measure.

Parameter

col	Name of field to filter
N	The number of top or bottom groups to filter

Example

```
Chart1.bindingInfo.setTopN("Company",3)
```



See Also

bindingInfo.setTopN(groupCol, n), for the equivalent Table function.

BINDINGINFO.ISTOPNREVERSE(COL)

Returns the state of the Top-N/Bottom-N filter.

Parameter

col	Name of the filtered field

Return

Boolean	true: filter is bottom N

```
                false: filter is top N
```

Example

```
var rev = Chart1.bindingInfo.isTopNReverse("Company");
```

See Also

bindingInfo.isTopNReverse(groupCol), for the equivalent Table function.

BINDINGINFO.SETTOPNREVERSE(COL,BOOLEAN)

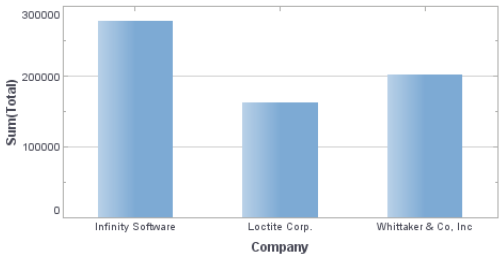
Specifies whether filter should extract the top-N or the bottom-N groups based on an aggregate.

Parameter

col	Name of field to filter
Boolean	true: filter bottom N false: filter top N

Example

```
Chart1.bindingInfo.setTopN("Company",3)
Chart1.bindingInfo.setTopNReverse("Company",true);
// Returns bottom-N groups
```



See Also

bindingInfo.setTopNReverse(groupCol, boolean), for the equivalent Table function.

BINDINGINFO.GETTOPNSUMMARYCOL(COL)

Returns the aggregate used for Top-N/Bottom-N filtering on a specified field.

Parameter

col	Name of filtered field

Example

```
var agg = Chart1.bindingInfo.getTopNSummaryCol("Company")
```

See Also

bindingInfo.getTopNSummaryCol(groupCol), for the equivalent Table function.

BINDINGINFO.SETTOPNSUMMARYCOL(COL,AGG)

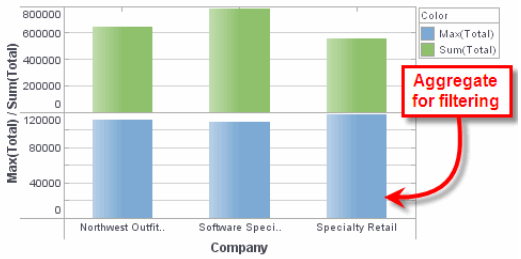
Specifies the aggregate to be used for Top-N/Bottom-N filtering on a given field.

Parameter

col	Name of field to filter
agg	The aggregate by which to filter

Example

```
Chart1.bindingInfo.setTopN("Company",3)
Chart1.bindingInfo.setTopNSummaryCol("Company","Max(Total)"
);
```



See Also

bindingInfo.setTopNSummaryCol(groupCol, col), for the equivalent Table function.

BINDINGINFO.GETFORMULA(COL,BINDING)

Returns the aggregation method used for a measure. The measure can be bound to the X or Y axis, or to one of the VisualFrames (*ColorFrame*, *ShapeFrame*, *SizeFrame*, and *TextureFrame*).

Parameter

col	Field for which to obtain summary method
binding	The field's binding (*Chart Constants* constant):

```
Chart.BINDING_FIELD: bound to X or Y axis
Chart.AESTHETIC_COLOR: bound to Color
Chart.AESTHETIC_SHAPE: bound to Shape
Chart.AESTHETIC_SIZE: bound to Size
Chart.AESTHETIC_TEXT: bound to Text
```

Return

The formula name (String)

Example

```
// For a field "Total" bound to the Y-axis:
var formula =
Chart1.bindingInfo.getFormula("Total",Chart.BINDING_FIELD);
```

See Also

bindingInfo.getFormula(i), for the equivalent Table function.

BINDINGINFO.SETFORMULA(COL, FORMULA, BINDING)

Sets the aggregation method for a measure (chart series). The measure can be bound to the X or Y axis, or to one of the VisualFrames (*ColorFrame*, *ShapeFrame*, *SizeFrame*, and *TextureFrame*).

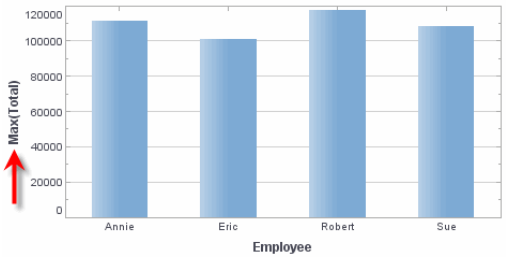
Parameter

col The name of the field to be summarized
formula A *Chart Constants* summarization formula (See *Summarization Formulas*.)
binding The field's binding (*Chart Constants* constant):

```
Chart.BINDING_FIELD: bound to X or Y axis
Chart.AESTHETIC_COLOR: bound to Color
Chart.AESTHETIC_SHAPE: bound to Shape
Chart.AESTHETIC_SIZE: bound to Size
Chart.AESTHETIC_TEXT: bound to Text
```

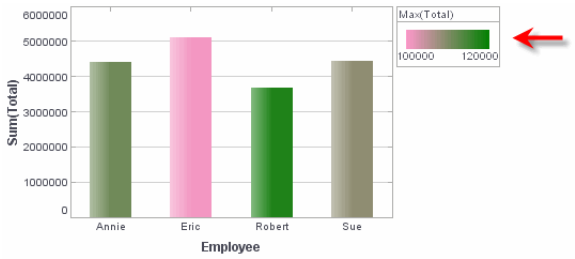
Example (axis-bound field)

```
// For a field "Total" bound to the Y-axis:
Chart1.bindingInfo.setFormula("Total", Chart.MAX_FORMULA,
Chart.BINDING_FIELD);
```



Example (frame-bound field)

```
// For a field "Total" bound to the ColorFrame:
Chart1.bindingInfo.setFormula("Total", Chart.MAX_FORMULA,
Chart.AESTHETIC_COLOR);
```



See Also

bindingInfo.setFormula(col, formula), for the equivalent Table function.

BINDINGINFO.GETPERCENTAGETYPE(COL,BINDING)

Returns the basis for computing percentages.

Parameter

col Field for which to obtain percentage type
binding The field's binding (*Chart Constants* constant):

- Chart.BINDING_FIELD: bound to X or Y axis
- Chart.AESTHETIC_COLOR: bound to Color
- Chart.AESTHETIC_SHAPE: bound to Shape
- Chart.AESTHETIC_SIZE: bound to Size
- Chart.AESTHETIC_TEXT: bound to Text

Return

Number

- 2 = PERCENTAGE_OF_GRANDTOTAL
- 1 = PERCENTAGE_OF_GROUP
- 0 = Chart.PERCENTAGE_NONE

Example

```
var type = Chart1.bindingInfo.getPercentageType("Total",
Chart.BINDING_FIELD)
```

BINDINGINFO.SETPERCENTAGETYPE(COL,TYPE)

Specifies the basis for computing percentages, grand total or group total.

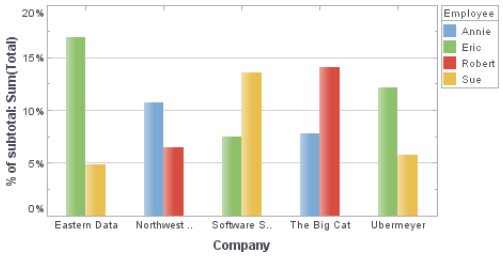
Parameter

col Name of field for which to set percentage
type *Chart Constants* constant:

- Chart.PERCENTAGE_NONE
- Chart.PERCENTAGE_OF_GRANDTOTAL
- Chart.PERCENTAGE_OF_GROUP

Example

```
Chart1.bindingInfo.setPercentageType("Total",
Chart.PERCENTAGE_OF_GROUP)
```



BINDINGINFO.GETSECONDARYFIELD(COL, BINDING)

Get the secondary field used in a bivariate aggregation method (e.g., correlation, covariance, etc.).

Parameter

col Field for which to obtain secondary field
binding Field binding (*Chart Constants* constant):

- Chart.BINDING_FIELD: bound to X or Y axis
- Chart.AESTHETIC_COLOR: bound to Color
- Chart.AESTHETIC_SHAPE: bound to Shape
- Chart.AESTHETIC_SIZE: bound to Size
- Chart.AESTHETIC_TEXT: bound to Text

Return

The field name (String)

Example

```
// For primary field "Price" bound to the Y-axis:
var col2 = Chart1.bindingInfo.getSecondaryField("Price",
Chart.BINDING_FIELD);
```

BINDINGINFO.SETSECONDARYFIELD(COL1, COL2, BINDING)

Set the secondary field to be used in a bivariate aggregation method (e.g., correlation, covariance, etc.).

Parameter

col1 Name of existing primary field
col2 Name of secondary field
binding Field binding (*Chart Constants* constant):
Chart.BINDING_FIELD: bound to X or Y axis
Chart.AESTHETIC_COLOR: bound to Color
Chart.AESTHETIC_SHAPE: bound to Shape
Chart.AESTHETIC_SIZE: bound to Size
Chart.AESTHETIC_TEXT: bound to Text

Example

```
// For primary field "Price" bound to the ColorFrame:
var formula = Chart1.bindingInfo.setSecondaryField("Price",
"Discount", Chart.AESTHETIC_COLOR);
```

BINDINGINFO.GETCOLORFIELD()

Returns the field bound to the *ColorFrame* (color coding).

Returns

String Field name used for color-coding

Example

```
var cfield = Chart1.bindingInfo.getColorField();
```

See Also

colorField, to return the runtime 'Color' binding.

BINDINGINFO.SETCOLORFIELD(COL, TYPE)

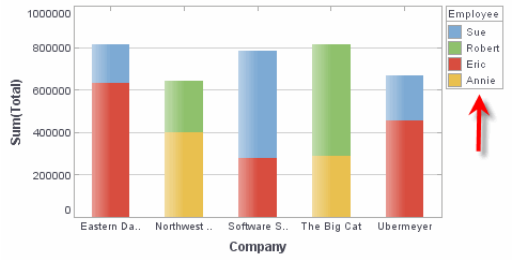
Bind the specified field to a *ColorFrame* (color coding).

Parameter

col Field to be used for color-coding
type *Chart Constants* constant: STRING, DATE, or NUMBER

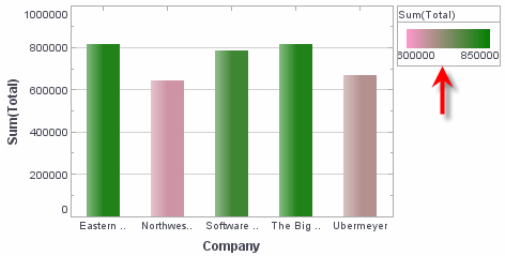
Example (Dimension field)

```
// For a dimension (categorical) field:
Chart1.bindingInfo.setColorField("Employee",Chart.STRING);
```



Example (Measure field)

```
// For a measure (numerical) field:
Chart1.bindingInfo.setColorField("Total",Chart.NUMBER);
```



See Also

bindingInfo.colorFrame, for information on the different *ColorFrame*s available.

BINDINGINFO.GETSHAPEFIELD()

Returns the field bound to the *ShapeFrame* (shape coding).

Returns

String	Field name used for shape-coding

Example

```
var sfield = Chart1.bindingInfo.getShapeField();
```

BINDINGINFO.SETSHAPEFIELD(COL, TYPE)

Bind the specified field to a *ShapeFrame* (shape coding). The aesthetic treatment applied to the chart elements depends on the type of the chart:

- Bar-type charts apply a *TextureFrame*, which draws different fill patterns. (For dimension binding, the default is the *CategoricalTextureFrame*. For measure binding, the default is *LeftTiltTextureFrame*.)
- Line-type charts apply a *LineFrame*, which draws different line styles. (For dimension binding, the default is the *CategoricalLineFrame*. For measure binding, the default is *LinearLineFrame*.)

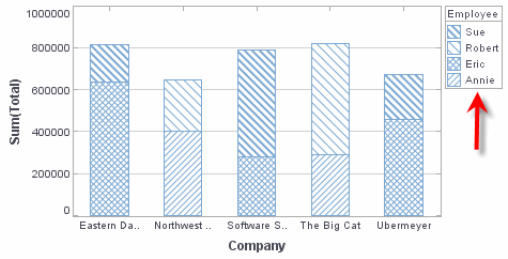
- Point-type charts apply a *ShapeFrame*, which draws different shape types. (For dimension binding, the default is the *CategoricalShapeFrame*. For measure binding, the default is *FillShapeFrame*.)

Parameter

col	Field to be used for shape-coding
type	*Chart Constants* constant: STRING, DATE, or NUMBER

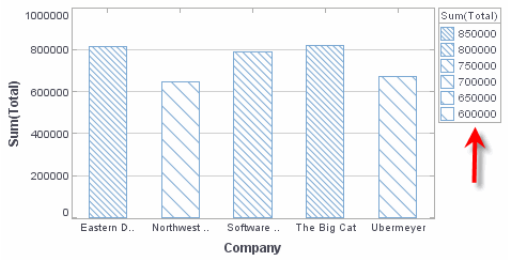
Example (Dimension field)

```
// For a dimension (categorical) field:
Chart1.bindingInfo.setShapeField("Employee",Chart.STRING);
```



Example (Measure field)

```
// For a measure (numerical) field:
Chart1.bindingInfo.setShapeField("Total",Chart.NUMBER);
```



See Also

bindingInfo.textureFrame, for information on setting the *TextureFrame*.
bindingInfo.lineFrame, for information on setting the *LineFrame*.
bindingInfo.shapeFrame, for information on setting the *ShapeFrame*.

BINDINGINFO.GETSIZEFIELD()

Returns the field bound to the *SizeFrame* (size coding).

Returns

String	Field name used for size-coding

Example

```
var sfield = Chart1.bindingInfo.getSizeField();
```

BINDINGINFO.SETSIZEFIELD(COL, TYPE)

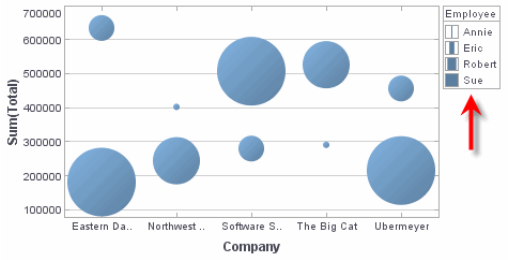
Bind the specified field to a *SizeFrame* (size coding).

Parameter

col	Field to be used for size-coding
type	*Chart Constants* constant: STRING, DATE, or NUMBER

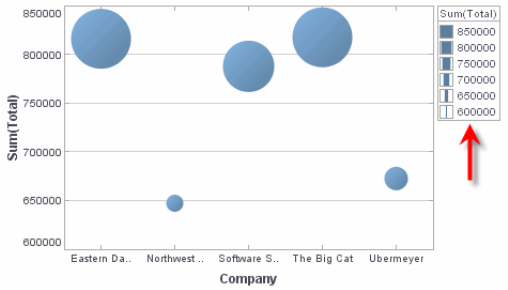
Example (Dimension field)

```
// For a dimension (categorical) field:
Chart1.bindingInfo.setSizeField("Employee",Chart.STRING);
```



Example (Measure field)

```
// For a measure (numerical) field:
Chart1.bindingInfo.setSizeField("Total",Chart.NUMBER);
```



See Also

bindingInfo.sizeFrame, for information on the different *ShapeFrame*s available.

BINDINGINFO.GETTEXTFIELD()

Returns the field bound to the *TextFrame* (text coding).

Returns

String	Field name used for text-coding

Example

```
var tfield = Chart1.bindingInfo.getTextField();
```

BINDINGINFO.SETTEXTFIELD(COL, TYPE)

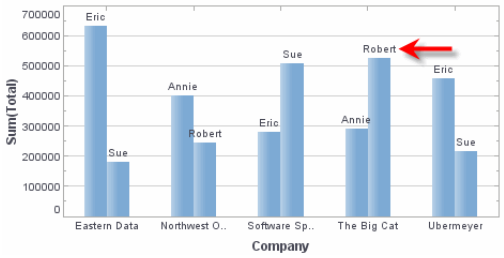
Bind the specified field to a *TextFrame* (text coding).

Parameter

col Field to be used for text-coding
type *Chart Constants* constant: STRING, DATE, or NUMBER

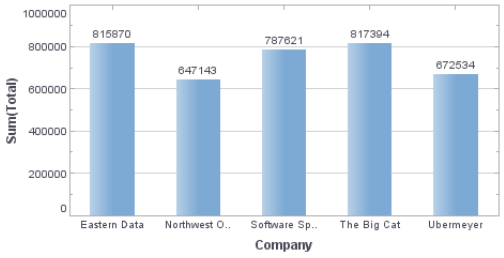
Example (Dimension field)

```
// For a dimension (categorical) field:
Chart1.bindingInfo.setTextField("Employee",Chart.STRING);
```



Example (Measure field)

```
// For a measure (numerical) field:
Chart1.bindingInfo.setTextField("Total",Chart.NUMBER);
```



BINDINGINFO.SETCANDLEBINDINGFIELD(ARR)

Configure the binding for a candlestick-type chart. A candlestick chart displays four different measures, “low,” “high,” “opening,” and “closing,” and is most often used to plot trading information. The parameter `arr` contains information for plotting one of these four measures.

Parameter

arr Array of properties for a candlestick chart measure:

`[field1,measure,formula,option,field2]`

The items in the array are described below:

field1 Field containing data corresponding to **measure**.
measure The measure represented by **field1**:

```
                Chart.HIGH: field1 defines the high values
                Chart.LOW: field1 defines the low values
                Chart.OPEN: field1 defines the opening values
                Chart.CLOSE: field1 defines the closing
values
formula         A Chart Constants summarization formula
                (See Summarization Formulas.)
option          Specifies how percentages should be computed:
                       Chart.PERCENTAGE_NONE
                       Chart.PERCENTAGE_OF_GRANDTOTAL
                       Chart.PERCENTAGE_OF_GROUP
field2          The second operand for a bivariate formula
(e.g., correlation).
```

The first two items in the array (`field1`, `measure`) are required. The others are optional.

Example

To create a candlestick chart that displays the average high, low, opening, and closing prices by quarter, follow the steps below:

1. Create a new report or Viewsheet. For a Viewsheet, select the 'Stock History' > 'Stock Prices' query as the data source.

2. Add a new Chart element to the report or Viewsheet.

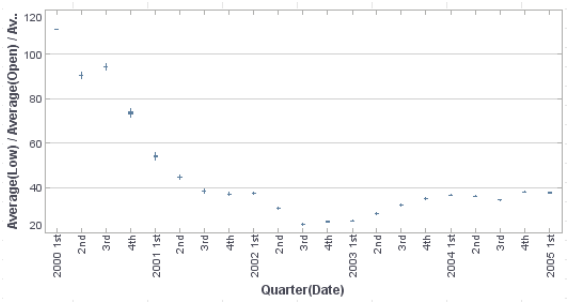
3. Enter the following script in the onLoad or onRefresh handler of the report or Viewsheet.

```
Chart1.query='Stock Prices';

Chart1.chartStyle = Chart.CHART_CANDLE;
var data_high = ["Stock
Prices.High",Chart.HIGH,Chart.AVERAGE_FORMULA];
var data_low = ["Stock
Prices.Low",Chart.LOW,Chart.AVERAGE_FORMULA];
var data_close = ['Stock Prices.Close/
Last',Chart.CLOSE,Chart.AVERAGE_FORMULA];
var data_open = ['Stock
Prices.Open',Chart.OPEN,Chart.AVERAGE_FORMULA];

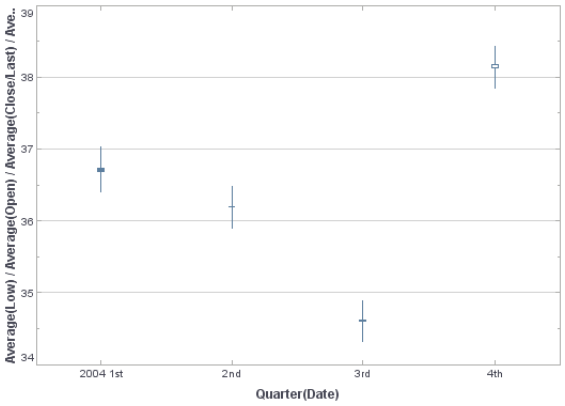
Chart1.bindingInfo.xFields=[['Date',Chart.DATE]];
Chart1.bindingInfo.setGroupOrder('Date',Chart.QUARTER_INTER
VAL);
Chart1.bindingInfo.setCandleBindingField(data_high);
Chart1.bindingInfo.setCandleBindingField(data_low);
Chart1.bindingInfo.setCandleBindingField(data_close);
Chart1.bindingInfo.setCandleBindingField(data_open);
```

4. Preview the chart.



In a Viewsheet, you can also use the 'Zoom' tool to limit the date range.

5. To observe the structure of the chart more closely, set a condition on the chart to limit the date range (e.g., limit to the year of 2004).



Note that for each quarter, the values of each of the measures (`High`, `Low`, `Close/Last`, `Open`) have been independently aggregated as specified by the corresponding formula parameter (in this case, `Chart.AVERAGE_FORMULA` for each).

The "high" and "low" measures are represented by the extremes of the candle "wick," and the "open" and "close" measures are represented by the extremes of the candle "body." Candles for which the "open" measure exceeds the value of the "close" measure are shown filled. Candles for which the "close" measure exceeds the value of the "open" measure are shown unfilled.

See Also

bindingInfo.getCandleBindingField(measure), to obtain current binding information.

BINDINGINFO.GETCANDLEBINDINGFIELD(MEASURE)

Return the field used as the specified measure on a candlestick-type chart, together with associated binding information.

Parameter

measure The measure to return:

```
                Chart.HIGH: measure defining the high values
                Chart.LOW: measure defining the low values
                Chart.OPEN: measure defining the opening
values
                Chart.CLOSE: measure defining the closing
values
```

Example

```
Chart1.bindingInfo.getCandleBindingField(Chart.LOW);
// returns (for example): lowPrice[lowPrice, null, Average,
0]
```

See Also

bindingInfo.setCandleBindingField(arr), to set the binding information.

BINDINGINFO.SETSTOCKBINDINGFIELD(ARR)

Configure the binding for a stock-type chart, also known as an open-hi-low-close (OHLC) chart. A stock chart displays four different measures, “low,” “high,” “opening,” and “closing,” and is most often used to plot trading information. The parameter `arr` contains information for plotting one of these four measures.

Parameter

arr	Array of properties for a stock chart measure: [field1,measure,formula,option,field2]

The items in the array are described below:

field1	Field containing data coresponding to **measure**.
measure	The measure represented by **field1**: Chart.HIGH: **field1** defines the high values Chart.LOW: **field1** defines the low values Chart.OPEN: **field1** defines the opening values Chart.CLOSE: **field1** defines the closing values
formula	A *Chart Constants* summarization formula (See *Summarization Formulas*.)
option	Specifies how percentages should be computed: Chart.PERCENTAGE_NONE Chart.PERCENTAGE_OF_GRANDTOTAL Chart.PERCENTAGE_OF_GROUP
field2	The second operand for a bivariate **formula** (e.g., correlation).

The first two items in the array (`field1`, `measure`) are required. The others are optional.

Example

To create a stock chart that displays the average high, low, opening, and closing prices by quarter, follow the steps below:

1. Create a new report or Viewsheet. For a Viewsheet, select the ‘Stock History’ > ‘Stock Prices’ query as the data source.

2. Add a new Chart element to the report or Viewsheet.

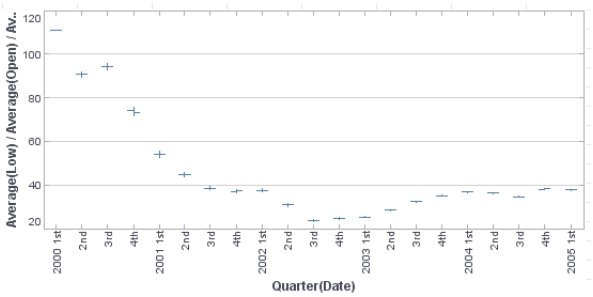
3. Enter the following script in the onLoad or onRefresh handler of the report or Viewsheet.

```
Chart1.query='Stock Prices';

Chart1.chartStyle = Chart.CHART_STOCK;
var data_high = ["Stock
Prices.High",Chart.HIGH,Chart.AVERAGE_FORMULA];
var data_low = ["Stock
Prices.Low",Chart.LOW,Chart.AVERAGE_FORMULA];
var data_close = ['Stock Prices.Close/
Last',Chart.CLOSE,Chart.AVERAGE_FORMULA];
var data_open = ['Stock
Prices.Open',Chart.OPEN,Chart.AVERAGE_FORMULA];

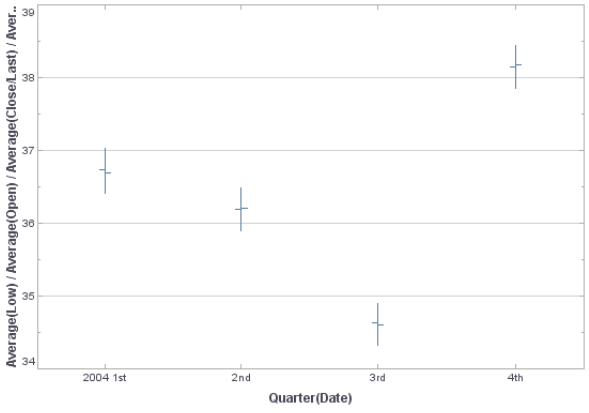
Chart1.bindingInfo.xFields=[['Date',Chart.DATE]];
Chart1.bindingInfo.setGroupOrder('Date',Chart.QUARTER_INTER
VAL);
Chart1.bindingInfo.setStockBindingField(data_high);
Chart1.bindingInfo.setStockBindingField(data_low);
Chart1.bindingInfo.setStockBindingField(data_close);
Chart1.bindingInfo.setStockBindingField(data_open);
```

4. Preview the chart.



In a Viewsheet, you can also use the 'Zoom' tool to limit the date range.

5. To observe the structure of the chart more closely, set a condition on the chart to limit the date range (e.g., limit to the year of 2004).



Note that for each quarter, the values of each of the measures (`High`, `Low`, `Close/Last`, `Open`) have been independently aggregated as specified by the corresponding formula parameter (in this case, `Chart.AVERAGE_FORMULA` for each).

The “high” and “low” measures are represented by the extremes of the vertical line, and the “open” and “close” measures are represented, respectively, by the left and right horizontal lines.

See Also

bindingInfo.getStockBindingField(measure), to obtain current binding information.

BINDINGINFO.GETSTOCKBINDINGFIELD(MEASURE)

Return the field used as the specified measure on a stock-type (OHLC) chart, together with associated binding information.

Parameter

measure The measure to return:
Chart.HIGH: measure defining the high values
Chart.LOW: measure defining the low values
Chart.OPEN: measure defining the opening values
Chart.CLOSE: measure defining the closing values

Example

```
Chart1.bindingInfo.getStockBindingField(Chart.LOW);
// returns (for example): lowPrice[lowPrice, null, Average, 0]
```

See Also

bindingInfo.setStockBindingField(arr), to set the binding information.

BINDINGINFO.COLORFRAME

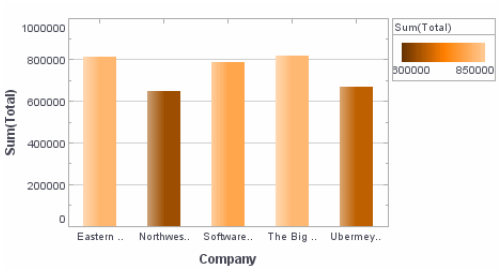
Specifies the aesthetic color treatment for the chart elements. A *ColorFrame* can color-code chart elements by value, or simply specify a static color scheme. The default for dimensions is *CategoricalColorFrame*, and the default for measures is *GradientColorFrame*.

Type

ColorFrame

Example

```
Chart1.bindingInfo.setColorField("Total",Chart.NUMBER);
Chart1.bindingInfo.colorFrame = new HeatColorFrame;
```



See Also

Modifying a Chart Element using API Functions, to apply a *ColorFrame* outside of the data binding.
bindingInfo.colors, to set a static color for a particular measure.
bindingInfo.setColorField(col, type), for information on binding a color field.

BINDINGINFO.SHAPEFRAME

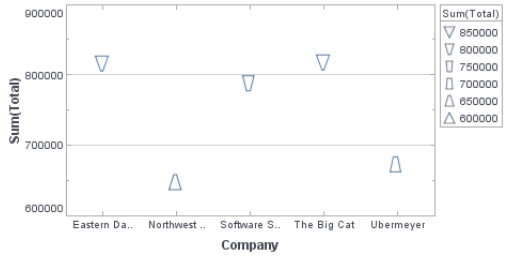
Specifies the aesthetic treatment for shape-coded data on point-type charts. A *ShapeFrame* can shape-code chart elements by value, or simply specify a static shape style.

Type

ShapeFrame (point-type charts)

Example

```
Chart1.bindingInfo.setShapeField("Total",Chart.NUMBER);
Chart1.bindingInfo.shapeFrame = new TriangleShapeFrame;
```



See Also

Modifying a Chart Element using API Functions, to apply a *ShapeFrame* outside of the data binding.
bindingInfo.shapes, to set a static shape for a particular measure.
bindingInfo.setShapeField(col, type), for information on binding a shape field.

BINDINGINFO.SIZEFRAME

Specifies the size of graphical elements. A *SizeFrame* can size-code chart elements by value, or simply specify a static size. The default for

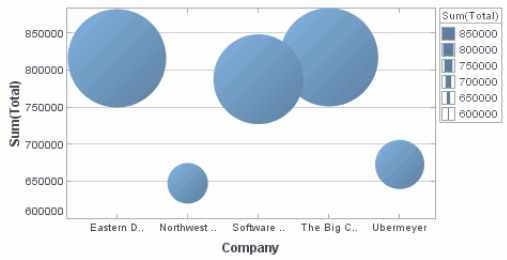
dimensions is *CategoricalSizeFrame*. The default for measures is *LinearSizeFrame*.

Type

SizeFrame

Example

```
Chart1.bindingInfo.setSizeField("Total",Chart.NUMBER);
Chart1.bindingInfo.sizeFrame = new LinearSizeFrame;
Chart1.bindingInfo.sizeFrame.smallest = 10;
Chart1.bindingInfo.sizeFrame.largest = 50;
Chart1.bindingInfo.sizeFrame.max = 100;
```



See Also

Modifying a Chart Element using API Functions, to apply a *SizeFrame* outside of the data binding.
bindingInfo.size, to set a static size .
bindingInfo.setSizeField(col, type), for information on binding a size field.

BINDINGINFO.TEXTUREFRAME

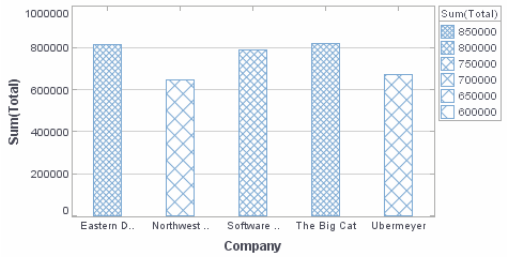
Specifies the aesthetic treatment (fill pattern) for the shape-coded data on a bar-type chart. A *TextureFrame* can texture-code chart elements by value, or simply specify a static texture style.

Type

TextureFrame (bar-type chart)

Example

```
Chart1.bindingInfo.setShapeField("Total",Chart.NUMBER);
Chart1.bindingInfo.textureFrame = new GridTextureFrame;
```



See Also

Modifying a Chart Element using API Functions, to apply a *TextureFrame* outside of the data binding.
bindingInfo.textures, to set a static texture for a particular measure.
bindingInfo.setShapeField(col, type), for information on binding a shape field.

BINDINGINFO.LINEFRAME

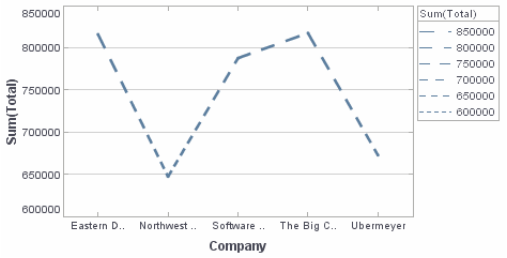
Specifies the aesthetic treatment (line style) for the shape-coded data on a line-type chart. A *LineFrame* can texture-code chart elements by value, or simply specify a static texture style.

Type

LineFrame (line-type chart)

Example

```
Chart1.bindingInfo.setShapeField("Total",Chart.NUMBER);
Chart1.bindingInfo.lineFrame = new LinearLineFrame;
```



See Also

Modifying a Chart Element using API Functions, to apply a *LineFrame* outside of the data binding.
bindingInfo.lines, to set a static texture for a particular measure.
bindingInfo.setShapeField(col, type), for information on binding a shape field.

BINDINGINFO.COLORS

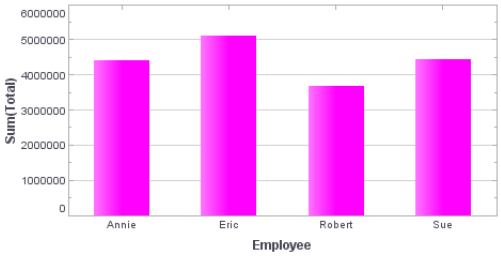
Specifies a *StaticColorFrame* (fixed color) for a particular measure.

Type

StaticColorFrame

Example

```
Chart1.bindingInfo.colors['Sum(Total)'] = new
StaticColorFrame(java.awt.Color(0xFF00FF));
```



See Also

Modifying a Chart Element using API Functions, to apply a *ColorFrame* outside of the data binding.

bindingInfo.colorFrame, for information on assigning value-keyed frames.

BINDINGINFO.SHAPES

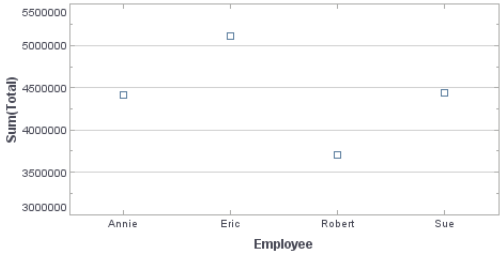
Specifies a *StaticShapeFrame* (fixed shape) for a particular measure on point-type charts.

Type

StaticShapeFrame (point-type charts)

Example

```
Chart1.bindingInfo.shapes['Sum(Total)'] = new
StaticShapeFrame(GShape.SQUARE);
```



See Also

Modifying a Chart Element using API Functions, to apply a *ShapeFrame* outside of the data binding.

bindingInfo.shapeFrame, for information on assigning value-keyed frames.

BINDINGINFO.SIZE

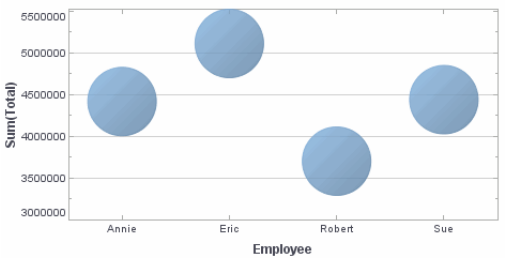
Specifies a *StaticSizeFrame* (fixed size) for all measures.

Type

StaticSizeFrame

Example

```
Chart1.bindingInfo.size = new StaticSizeFrame(30);
```



See Also

Modifying a Chart Element using API Functions, to apply a *SizeFrame* outside of the data binding.

bindingInfo.sizeFrame, for information on assigning value-keyed frames.

BINDINGINFO.TEXTURES

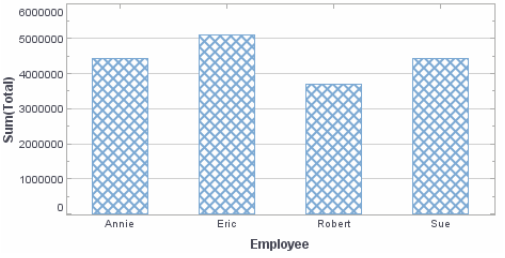
Specifies a *StaticTextureFrame* (fixed pattern) for a particular measure on bar-type charts.

Type

`StaticTextureFrame` (bar-type charts)

Example

```
Chart1.bindingInfo.textures['Sum(Total)'] = new
StaticTextureFrame(GTexture.PATTERN_5);
```



See Also

Modifying a Chart Element using API Functions, to apply a *TextureFrame* outside of the data binding.

bindingInfo.textureFrame, for information on assigning value-keyed frames.

BINDINGINFO.LINES

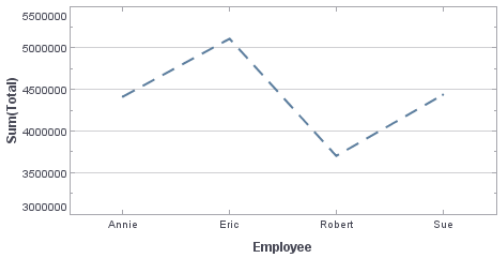
Specifies a *StaticLineFrame* (fixed line style) for a particular measure on line-type charts.

Type

`StaticLineFrame` (line-type charts)

Example

```
Chart1.bindingInfo.lines['Sum(Total)'] = new
StaticLineFrame(GLine.LARGE_DASH);
```



See Also

Modifying a Chart Element using API Functions, to apply a *LineFrame* outside of the data binding.
bindingInfo.lineFrame, for information on assigning value-keyed frames.

BINDINGINFO.ADDMAPPING(COL, VALUE, GEOCODE)

Specifies the mapping between a `value` of field `col` and a defined geographical code, `geocode`. The geographical codes are defined in the CSV files within *inetsoft/graph/geo/data* directory of the *bisuite.jar* file for the corresponding map.

Parameter

col	Field from which value should be mapped
value	Value to be mapped to a defined geo location
geocode	A geographical location in the geo data file

The latitude and longitude for code 'USA0113' are defined in inetsoft/graph/geo/data/us-cities.csv.

Example

```
// Map the value 'Queens' (in column 'city') to 'New York
City'.
Chart1.bindingInfo.addMapping('city','Queens','USA0113');
```

See Also

bindingInfo.getMappings(col), to retrieve the currently specified mappings.
bindingInfo.removeMapping(col,value), to remove a currently specified mapping.

BINDINGINFO.GETMAPPINGS(COL)

Returns the mappings that exist between the values in field `col` and defined geographical codes (as a two dimensional string array).

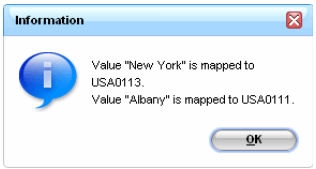
Parameter

col	Field whose mappings should be returned

Example

```
// Display mappings in column 'city':
var mapArr = Chart1.bindingInfo.getMappings('city');
```

```
for(i=0 ; i<mapArr.length ; i++) {
  alert('Value "' + mapArr[i][0] + '" is mapped to ' +
mapArr[i][1] + '.');
}
```



See Also

bindingInfo.addMapping(col, value, geocode), to add a specified geographical mapping.
bindingInfo.removeMapping(col,value), to remove a currently specified mapping.

BINDINGINFO.REMOVEMAPPING(COL,VALUE)

Removes the mapping that exists between the specified `value` of field `col` and a defined geographical code.

Parameter

col	Field from which value is mapped
value	Value whose mapping should be removed

Example

```
// Remove mapping for value 'Queens' in column 'city':
Chart1.bindingInfo.removeMapping('city','Queens');
```

See Also

bindingInfo.addMapping(col, value, geocode), to add a specified geographical mapping.
bindingInfo.getMappings(col), to retrieve the currently specified mappings.

CR.2 Object Hierarchy

The figure below shows the object structure of the Style Chart API. Among the significant objects, *GraphElement* contains the elements that graphically represent data (lines, bars, etc.). *VisualFrame* contains information about mapping data dimensions to physical properties (size, color, etc.), and *Scale* contains the scaling information for such mappings. *GraphForm* contains information for manually-drawn chart objects.



CR.3 Getting and Setting Chart Properties

The Chart API provides “getter” and “setter” methods that allow you to read and write most chart properties from script. The following sections provide definitions and examples for the “setter” methods.

In general, you can call the “getter” method corresponding to a particular “setter” method by simply changing the method name prefix.

For example, the “getter” method corresponding to `LabelForm.setAlignmentY()` is `LabelForm.getAlignmentY()`. The following script illustrates how you can “get” the alignment of one chart component, and use it to “set” the alignment of another.

```
dataset = [["State","Quantity"], ["NJ",200], ["NY",300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var form1 = new LabelForm();
var form2 = new LabelForm();
form1.setLabel("label1");
form1.setPoint(new java.awt.Point(100, 100));
```

```
form1.setAlignmentY(Chart.TOP_ALIGNMENT);
form2.setLabel("label2");
form2.setPoint(new java.awt.Point(200, 100));
form2.setAlignmentY(form1.getAlignmentY());
graph.addForm(form1);
graph.addForm(form2);
graph.addElement(elem);
```

Many attributes can also be accessed as object properties. For example:

```
form2.alignmentY = form1.alignmentY;
```

CR.4 General Chart Properties

The following sections present the top-level components required to build a chart. These include the *DataSet* object, which encapsulates data from a JavaScript array, and the *EGraph* object, which represents the global chart object. The *TextSpec*, *AxisSpec*, *LegendSpec*, and *TitleSpec* objects are used by various other components to set formatting for text, axes, legends, and titles.

CR.4.1 Data

The Chart `data` object is a two-dimensional array containing the aggregate data displayed on the Chart. You can use standard array notation, `data[i][j]`, to access the data value in row `i` and column `j`. Use `data.length` and `data.size`, respectively, to obtain the number of X-axis labels and number of datasets. For an example, see *Accessing Chart Data in Script* in *Chart Script Tutorial*.

Tip: You can use the Chart's "query" property to set a query in the onLoad Handler. See *Modifying a Chart Data Binding* for an example.

You can assign the results of a runQuery() function to the Data object to set the chart dataset:

```
data = runQuery("sales by state");
```

The Data object also supports formula table syntax such as the following:

```
// Data in aggregated measure/column 'Sum(Sales)'
data["Sum(Sales)"]

// Data in 'Sum(Sales)' for state of NJ
data["Sum(Sales)@State:NJ"]

// Data in 'Sum(Sales)' where the value exceeds 1000000
data["State?Sum(Sales) > 1000000"]
```

See Also

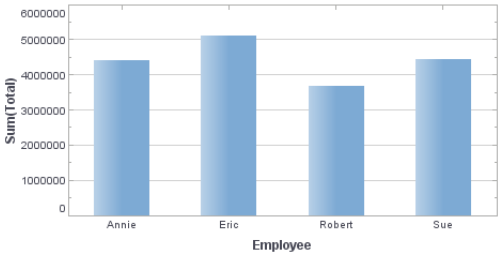
Creating a Chart Using API Functions, for an example using runQuery().
DataSet, for information on setting the chart dataset.
Table, for information on accessing the dataset along with column headers.

CR.4.2 Table

The Chart's Table object is a two-dimensional array containing the aggregates records (summarized data) bound to a chart. You can use standard array notation, `table[i][j]`, to access the data value in row `i` and column `j`. Use `table.length` and `table.size`, respectively, to obtain the number of rows and columns in the summarized data.

The values provided by the Table object are the same as those provided by the *DataSet* object, except that the first row, `table[0][j]`, contains the column headers rather than the first row of data.

For example, consider the following chart bound to the 'All Sales' query.



Example

```
// First row of 'table' has headers: Employee, Sum(Total)
var str = "";
for (var j = 0; j < table.size; j++) {
  str += table[0][j] + ", ";
}
log(str);
```

Example

```
// First row of 'dataset' has data: Annie, 4412983.0
var str = "";
for (var j = 0; j < dataset.getColCount(); j++) {
  str += dataset.getData(j,0) + ", ";
}
log(str);
```

See Also

DataSet, for information on accessing the chart's aggregated dataset.
Data, for information on accessing Chart data with advanced syntax.

CR.4.3 DataSet

The DataSet object allows you to set the values to be displayed on the graph. It has the form of a two-dimensional array, where each column represents a distinct measure. You can specify the DataSet by one of the following methods:

- In Style Studio or Visual Composer, set the chart binding using the Chart Editor.

- In report or Viewsheet script, assign a JavaScript array to the `dataset` property.

```
dataset = [["State", "Quantity"],["NJ", 200],["NY",
300],["PA", 370],["CT", 75]];
```

See *Running a Query from Script* in *Advanced Dashboard Scripting* and *Running a Query from Script* in *Report Scripting* for details about runQuery().

- In report or Viewsheet script, assign a query result (e.g., 'sales by state') to the `dataset` property.

```
dataset = runQuery("sales by state");
```

The DataSet object is also accessible for reading. See the `getData()` method for more information. However, in many cases, the *Data* property provides more convenient access to Chart data.

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Setting the DataSet)

```
dataset = [["State","Quantity"], ["CA",200], ["NY",3000]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
graph.addElement(elem);
```

Full Name

```
inetsoft.graph.data.DataSet
```

See Also

Binding Data to a Chart in Script, for various approaches to binding data to a chart.

Creating a Chart Using API Functions, for report and Viewsheet chart datasets.

DataSet: getData(), getRowCount(), getColCount()

DataSet.GETDATA(COLUMN,ROW)

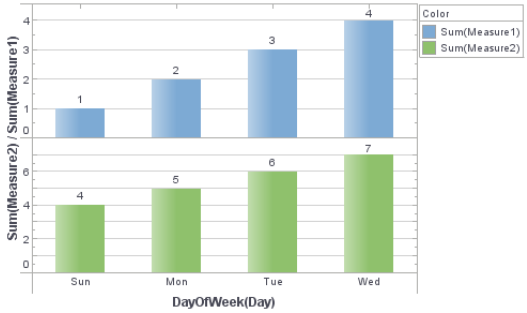
Returns the value in the *DataSet* object specified by the column and row indices. The first column (index 0) contains the X-axis labels.

Parameters

column	Column index of value to return
row	Row index of value to return

Note: In many cases, the *Data* property is more convenient to use than *DataSet*.

For example, consider a chart with two datasets (measures), as shown below:



Note the index order: (column, row).

Use `dataset.getData(j,i)` with row index `i` and column index `j` to access these plotted values. The *DataSet.getRowCount()* and *DataSet.getColCount()* functions provide the number of rows and columns of summarized data.

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example

```
var str = "";

// Loop through rows
for (var i = 0; i < dataset.getRowCount(); i++) {
   str = "";

   // Loop through columns
   for (var j = 0; j < dataset.getColCount(); j++) {
      str += dataset.getData(j, i) + ",";
   }

   // Output the results
   log(str);    // Style Studio
   alert(str);  // Visual Composer

}
```

The following output is written by the `log()` function (for Style Studio) to the Console window:

```
Sun May 31 00:00:00 EDT 2009, 1.0, 4.0,
Mon Jun 01 00:00:00 EDT 2009, 2.0, 5.0,
Tue Jun 02 00:00:00 EDT 2009, 3.0, 6.0,
Wed Jun 03 00:00:00 EDT 2009, 4.0, 7.0,
```

The values in the first column (e.g., Sun...2009, Mon...2009, etc.) are the dates on the X-axis, while the second and third columns represent the two datasets (Measure 1 and Measure 2).

See Also

Table, for information on accessing the dataset along with column headers.

Data, for information on accessing pre-aggregate data.

DataSet.GETROWCOUNT()

Returns the number of rows in the *DataSet* (including the header row).

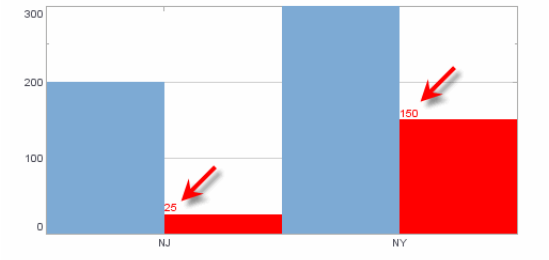
Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example

```
// Create a chart with two datasets:
dataset = [["State", "Total", "Profit"],["NJ", 200,
25],["NY", 300, 150]];
graph = new EGraph();
var elem1 = new IntervalElement("State", "Total");
var elem2 = new IntervalElement("State", "Profit");
var frame = new StaticColorFrame(java.awt.Color.red);
elem2.setColorFrame(frame);
graph.addElement(elem1);
graph.addElement(elem2);

// Loop through the rows of the second dataset (Profit),
// and place labels on the bars.
for (var i=0; i<dataset.getRowCount(); i++) {
  var form = new LabelForm();
  form.setLabel(dataset.getData(2,i));
  form.setValues([dataset.getData(0,i),
                  dataset.getData(2,i)]);
  var spec = new TextSpec();
  spec.setColor(java.awt.Color.red)
  form.setTextSpec(spec)
  graph.addForm(form)
}
```

Note: In many cases, the *Data* property is more convenient to use than *DataSet*.



See Also

DataSet.getData(column,row), to retrieve data from the *DataSet* object.

DataSet.GETCOLCOUNT()

Returns the number of columns in the *DataSet*.

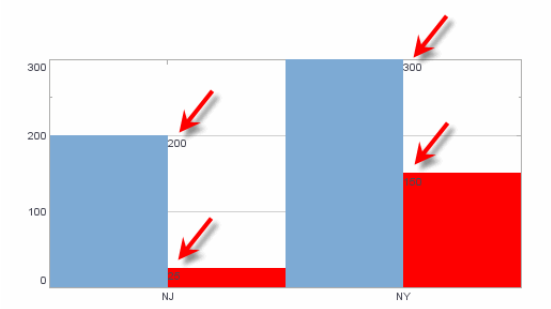
Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
// Create a chart with two datasets:
dataset = [["State", "Total", "Profit"],["NJ", 200, 25],
["NY", 300, 150]];
graph = new EGraph();
var elem1 = new IntervalElement("State", "Total");
var elem2 = new IntervalElement("State", "Profit");
var frame = new StaticColorFrame(java.awt.Color.red);
elem2.setColorFrame(frame);
graph.addElement(elem1);
graph.addElement(elem2);
```

Note: In many cases, the *Data* property is more convenient to use than *DataSet*.

```
// Loop through the rows and columns,
// and place labels on the bars.
for (var i=0; i<dataset.getRowCount(); i++) {
  for (var j=0; j<dataset.getColCount(); j++) {
    var form = new LabelForm();
    form.setColor(java.awt.Color.black);
    form.setLabel(dataset.getData(j,i));
    form.setValues([dataset.getData(0,i),
                    dataset.getData(j,i)-20]);
    graph.addForm(form)
  }
}
```



See Also

DataSet, to use a Data Block or query as the *dataset* source.
DataSet.getData(column,row), to retrieve data from the *DataSet* object.

DataSet.SETORDER(DIM,ARR)

Set a manual label ordering for a given dimension field.

Parameters

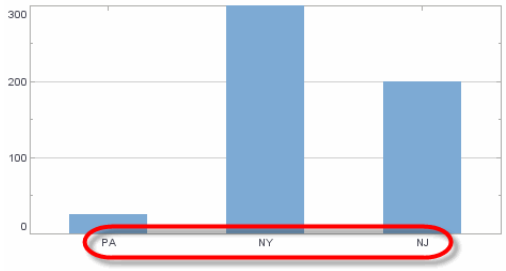
dim	String giving name of the dimension to sort
arr	Array of strings with desired label order: ['label1','label2','label3',...]

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"],["NJ", 200],["NY",
300],["PA", 25]];
```

```
dataset.setOrder('State',['PA','NY','NJ']);
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
graph.addElement(elem);
```



See Also

DataSet, to use a Data Block or query as the *dataset* source.
DataSet.getData(column,row), to retrieve data from the *DataSet* object.

CR.4.4 EGraph

The *EGraph* object represents the graph definition. To create a new EGraph object, call the *EGraph* constructor.

```
graph = new EGraph();
```

Full Name

`inetsoft.graph.EGraph`

See Also

Creating a Chart Using API Functions, for the basic steps in building a chart.

EGraph.ADDELEMENT(ELEM)

Adds the specified *GraphElement* to the chart. See *Chart Elements* for available elements.

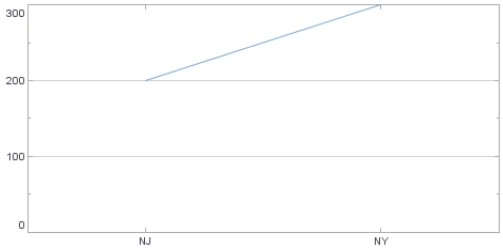
Parameter

elem a *GraphElement* object

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ", 200], ["NY", 300]];
graph = new EGraph();
var elem = new LineElement("State", "Quantity");
graph.addElement(elem);
```



See Also

DataSet, to use a Data Block or query as the *dataset* source.
Representing Multiple Measures, to add multiple elements.
Chart Elements, for available elements.

EGraph.ADDFORM(FORM)

Adds the specified *GraphForm* to the chart. See *Chart Annotation and Decoration* for available forms.

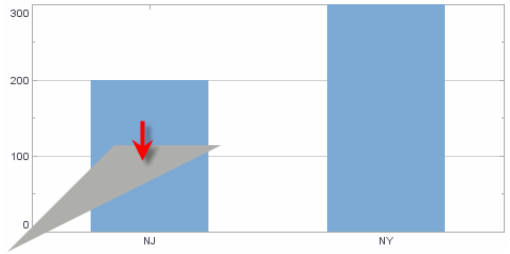
Parameter

form a *GraphForm* object

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ", 200], ["NY", 300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var form = new LineForm();
form.addPoint(new java.awt.Point(0,0));
form.addPoint(new java.awt.Point(100,100));
form.addPoint(new java.awt.Point(200,100));
form.setFill(true);
graph.addForm(form);
graph.addElement(elem);
```



See Also

DataSet, to use a Data Block or query as the *dataset* source.

EGraph.GETCOORDINATE()

Returns a handle to the graph's *Coordinate* object. This is useful in cases where the coordinates were not explicitly assigned to a variable at creation time.

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report)

Bind a bar-type chart to the sample 'All Sales' query, with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script:

```
var coord = graph.getCoordinate();
coord.transpose();
```

Example (Viewsheet)

Create a Viewsheet based on the 'Sales Explore' Worksheet. Bind a bar-type chart with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script:

```
var coord = graph.getCoordinate();
coord.transpose();
```

See Also

Appendix SC.12, *Changing the Chart Coordinates*, for information on modifying coordinates.

Appendix SC.6, *Chart Coordinates & Scaling*, for available coordinate sets.

EGraph.GETELEMENT(INDEX)

Returns a handle to the *GraphElement* object specified by the index. This is useful in cases where the element was not explicitly assigned to a variable at creation time.

Parameter

index Integer index of element

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report)

Bind a bar-type chart to the sample 'All Sales' query, with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script:

```
var elem = graph.getElement(0);
elem.setHint(GraphElement.HINT_SHINE,'false');
```

Example (Viewsheet)

Create a Viewsheet based on the 'Sales Explore' Worksheet. Bind a bar-type chart with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script:

```
var elem = graph.getElement(0);
elem.setHint(GraphElement.HINT_SHINE,'false');
```

See Also

Representing Multiple Measures, to add multiple elements.
Chart Elements, for available elements.

EGraph.GETELEMENTCOUNT()

Returns the number of *GraphElement* objects currently on the chart.

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State","Total 1","Total 2"], ["NJ",200,500],
["NY",300,400]];
graph = new EGraph();
graph.addElement(new LineElement("State","Total 1"));
graph.addElement(new LineElement("State","Total 2"));
var elemCount = graph.getElementCount();
for (var i=0;i<elemCount;i++) {
   graph.getElement(i).endArrow = true;
}
```

See Also

DataSet, to use a Data Block or query as the *dataset* source.

EGraph.GETFORM(INDEX)

Returns a handle to the *GraphForm* object specified by the index. This is useful in cases where the form was not explicitly assigned to a variable at creation time.

Parameter

index Integer index of form

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ", 200], ["NY", 300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
graph.addForm(new LineForm());
var form = graph.getForm(0);
```

```
form.addPoint(java.awt.Point(100,100));
form.addPoint(java.awt.Point(200,200));
form.setColor(java.awt.Color(0xff0000));
graph.addElement(elem);
```

See Also

DataSet, to use a Data Block or query as the *dataset* source.

Adding Decorative Elements, for information on manually adding elements.

Chart Annotation and Decoration, for available forms.

EGraph.GETFORMCOUNT()

Returns the number of forms currently present on the chart.

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ", 200], ["NY", 300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
graph.addForm(new LineForm());
graph.addForm(new LineForm());
graph.addForm(new LineForm());
var formCount = graph.getFormCount();
for (var i=0;i<formCount;i++) {
  graph.getForm(i).addValues(['NJ',i*100]);
  graph.getForm(i).addValues(['NY',300]);
  graph.getForm(i).setColor(java.awt.Color(0xff0000));
}
graph.addElement(elem);
```

See Also

DataSet, to use a Data Block or query as the *dataset* source.

EGraph.GETSCALE(FIELD)

Returns a handle to the *Scale* object specified by the field name parameter. This is useful in cases where the scale was not explicitly assigned to a variable at creation time.

Parameters

field String representing field name

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report)

Bind a bar-type chart to the sample ‘All Sales’ query, with ‘Company’ (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script:

```
var scale = graph.getScale("Sum(Total)");
scale.setMin(600000);
scale.setMax(1000000);
```

Example (Viewsheet)

Create a Viewsheet based on the 'Sales Explore' Worksheet. Bind a bar-type chart with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script:

```
var scale = graph.getScale("Sum(Total)");
scale.setMin(600000);
scale.setMax(1000000);
```

See Also

Changing Chart Scaling, for information on setting scales.
Chart Coordinates & Scaling, for available scales.

EGraph.SETLEGENDLAYOUT(VALUE)

Specifies the chart legend's position.

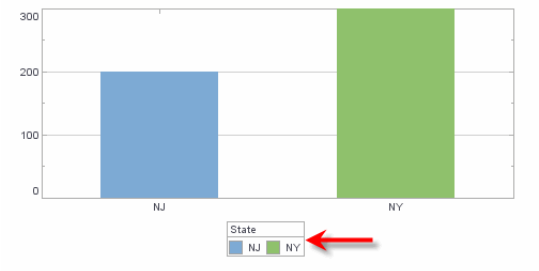
Parameter

value	Chart.BOTTOM	// below chart data
	Chart.TOP	// above chart data
	Chart.RIGHT	// right of chart
	Chart.LEFT	// left of chart
	Chart.IN_PLACE	// floated on chart
	Chart.NONE	// no legend

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ",200], ["NY",300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var frame = new CategoricalColorFrame();
frame.setField("State");
elem.setColorFrame(frame);
graph.setLegendLayout(Chart.BOTTOM);
graph.addElement(elem);
```



See Also

Legend Positions, for a list of available positions.
DataSet, to use a Data Block or query as the *dataset* source.

EGraph.SETCOORDINATE(COORD)

Sets the specified *Coordinate* for the chart. See *Chart Coordinates & Scaling* for available coordinate types.

Parameter

coord	a *Coordinate* object

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ", 200], ["NY", 300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var sscale = new CategoricalScale("State");
var qscale = new LinearScale("Quantity");
var coord = new RectCoord(sscale,qscale);
coord.transpose();
graph.setCoordinate(coord);
graph.addElement(elem);
```

See Also

DataSet, to use a Data Block or query as the *dataset* source.
Changing the Chart Coordinates, for information on modifying coordinates.
Chart Coordinates & Scaling, for available coordinate sets.

EGraph.SETSCALE(FIELD, SCALE)

Sets the specified *Scale* for the given axis. See *Chart Coordinates & Scaling* for available scales.

Parameters

field	name of axis to receive scale (String)
scale	scale to apply to axis (*Scale*)

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ", 200], ["NY", 300]];
graph = new EGraph();
var elem = new LineElement("State", "Quantity");
var qscale = new LinearScale("Quantity");
qscale.setMin(100);
qscale.setMax(500);
```

```
graph.addElement(elem);
graph.setScale("Quantity", qscale);
```

See Also

DataSet, to use a Data Block or query as the *dataset* source.
Changing Chart Scaling, for information on setting scales.
Chart Coordinates & Scaling, for available scales.

EGraph.SETXTITLESPEC(SPEC)

Specifies the X-axis title information for X-axis below the chart.

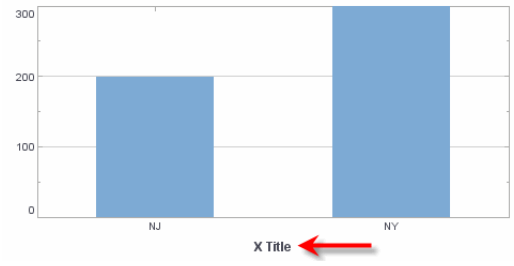
Parameter

spec a *TitleSpec* object

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ", 200], ["NY", 300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var spec = new TitleSpec();
spec.setLabel("X Title")
graph.setXTitleSpec(spec);
graph.addElement(elem);
```



See Also

DataSet, to use a Data Block or query as the *dataset* source.

EGraph.SETX2TITLESPEC(SPEC)

Specifies the X-axis title information for X-axis above the chart.

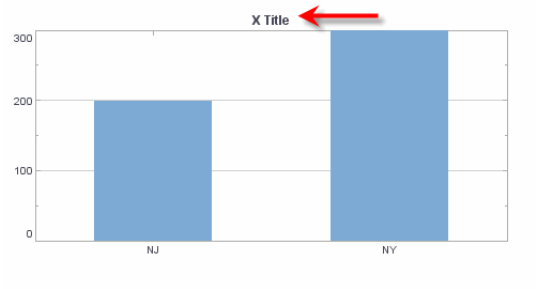
Parameter

spec a *TitleSpec* object

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ", 200], ["NY", 300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var spec = new TitleSpec();
spec.setLabel("X Title")
graph.setX2TitleSpec(spec);
graph.addElement(elem);
```



See Also

DataSet, to use a Data Block or query as the *dataset* source.

EGraph.SETYTITLESPEC(SPEC)

Specifies the Y-axis title information for Y-axis to left of chart.

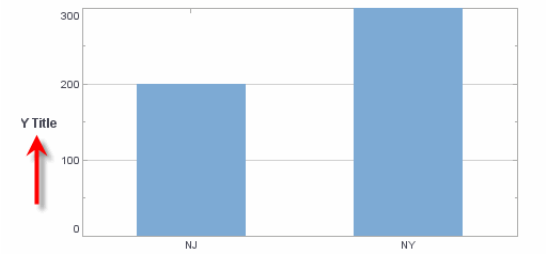
Parameter

spec a *TitleSpec* object

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ", 200], ["NY", 300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var spec = new TitleSpec();
spec.setLabel("Y Title")
graph.setYTitleSpec(spec);
graph.addElement(elem);
```



See Also

DataSet, to use a Data Block or query as the *dataset* source.

EGraph.SETY2TITLESPEC(SPEC)

Specifies the Y-axis title information for Y-axis to right of chart.

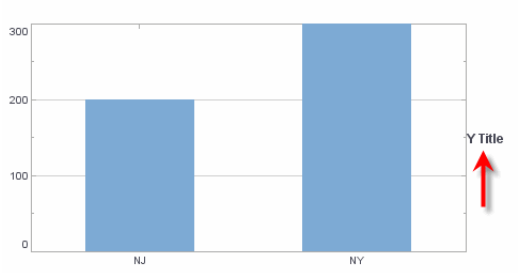
Type

spec a *TitleSpec* object

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ", 200], ["NY", 300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var spec = new TitleSpec();
spec.setLabel("Y Title")
graph.setY2TitleSpec(spec);
graph.addElement(elem);
```



See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.4.5 LegendSpec

The LegendSpec object contains legend formatting information.

Full Name

`inetsoft.graph.LegendSpec`

LegendSpec: setBackground, setBorder, setBorderColor, setPartial, setPosition, setPreferredSize, setTextFrame, setTextSpec, setTitle, setTitleTextSpec, setTitleVisible, setVisible

See Also

VisualFrame, to create groupings that generate a legend on the chart.
EGraph.setLegendLayout(value), to place the legend in a predefined location.

LegendSpec.SETBACKGROUND(VALUE)

Specifies the legend background color.

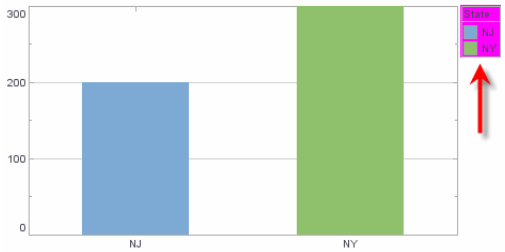
Parameter

value a `java.awt.Color` object

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ",200], ["NY",300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var frame = new CategoricalColorFrame();
frame.setField("State");
var spec = new LegendSpec();
spec.setBackground(java.awt.Color(0xff00ff));
frame.setLegendSpec(spec);
elem.setColorFrame(frame);
graph.addElement(elem);
```



See Also

DataSet, to use a Data Block or query as the *dataset* source.

LegendSpec.SETBORDER(VALUE)

Specifies the legend border style.

Parameter

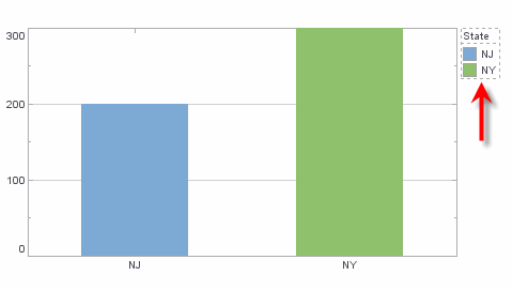
value a *GLine* object

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ",200], ["NY",300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var frame = new CategoricalColorFrame();
frame.setField("State");
var spec = new LegendSpec();
```

```
spec.setBorder(Chart.DASH_LINE);
frame.setLegendSpec(spec);
elem.setColorFrame(frame);
graph.addElement(elem);
```



See Also

DataSet, to use a Data Block or query as the *dataset* source.

LegendSpec.SETBORDERCOLOR(VALUE)

Specifies the legend border color.

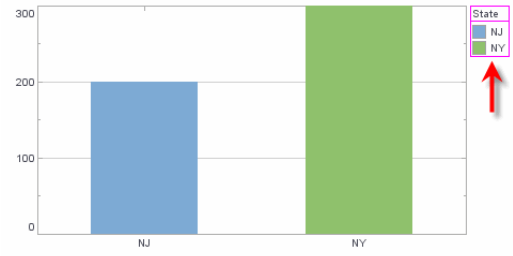
Parameter

value a `java.awt.Color` object

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ",200], ["NY",300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var frame = new CategoricalColorFrame();
frame.setField("State");
var spec = new LegendSpec();
spec.setBorderColor(java.awt.Color(0xff00ff));
frame.setLegendSpec(spec);
elem.setColorFrame(frame);
graph.addElement(elem);
```



See Also

DataSet, to use a Data Block or query as the *dataset* source.

LegendSpec.SETPARTIAL(BOOLEAN)

Specifies whether legend items can be ignored when there is insufficient space.

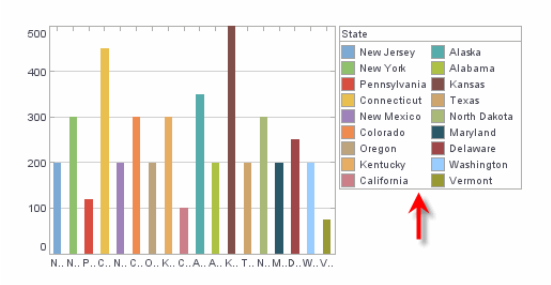
Type

boolean true: ignore items exceeding legend size
false: compress items to fit (default)

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["New Jersey",200],
           ["New York",300], ["Pennsylvania",120],
           ["Connecticut",450], ["New Mexico",200],
           ["Colorado",300], ["Oregon",200],
           ["Kentucky",300], ["California",100],
           ["Alaska",350], ["Alabama",200], ["Kansas",500],
           ["Texas",200], ["North Dakota",300],
           ["Maryland",200], ["Delaware",250],
           ["Washington",200], ["Vermont",75]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var frame = new CategoricalColorFrame();
frame.setField("State");
var spec = new LegendSpec();
spec.setPartial(true);
frame.setLegendSpec(spec);
elem.setColorFrame(frame);
graph.addElement(elem);
```



Drag the Chart handles to a make the chart smaller. Note how the contents of the legend are abridged.

See Also

DataSet, to use a Data Block or query as the *dataset* source.

LegendSpec.SETPOSITION(VALUE)

Specifies the position of the legend's bottom-left corner (in pixels or proportion) for cases when "in place" layout is used. (Positive values

specify distance from left/bottom. Negative values specify distance from right/top.)

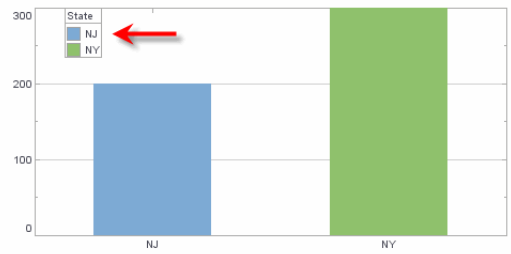
Type

value a subclass of java.awt.geom.Point2D
e.g.,
java.awt.Point for pixels
java.awt.geom.Point2D.Double for proportion

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ",200], ["NY",300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var frame = new CategoricalColorFrame();
frame.setField("State");
var spec = new LegendSpec();
spec.setPosition(java.awt.Point(50,245)); // in pixels
frame.setLegendSpec(spec);
elem.setColorFrame(frame);
graph.setLegendLayout(Chart.IN_PLACE);
graph.addElement(elem);
```



See Also

DataSet, to use a Data Block or query as the *dataset* source.
EGraph.setLegendLayout(value), to place the legend in a predefined location.

LegendSpec.SETPREFERREDSIZE(VALUE)

Specifies the legend size (pixels) for cases when *EGraph.setLegendLayout(value)* is set to `Chart.IN_PLACE`.

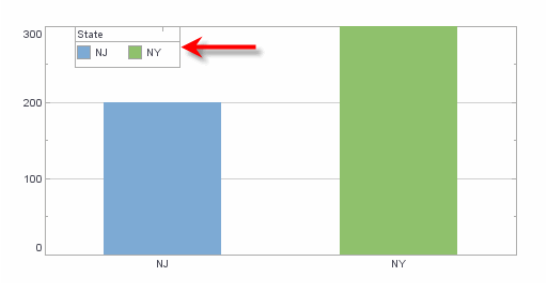
Parameter

value a java.awt.Dimension object

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ",200], ["NY",300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var frame = new CategoricalColorFrame();
frame.setField("State");
var spec = new LegendSpec();
spec.setPosition(java.awt.Point(50,245)); // in pixels
spec.setPreferredSize(java.awt.Dimension(100,40));
frame.setLegendSpec(spec);
elem.setColorFrame(frame);
graph.setLegendLayout(Chart.IN_PLACE);
graph.addElement(elem);
```



See Also

DataSet, to use a Data Block or query as the *dataset* source.

LegendSpec.SETTEXTFRAME(FRAME)

Specifies the *TextFrame* containing a mapping between legend values and replacement text.

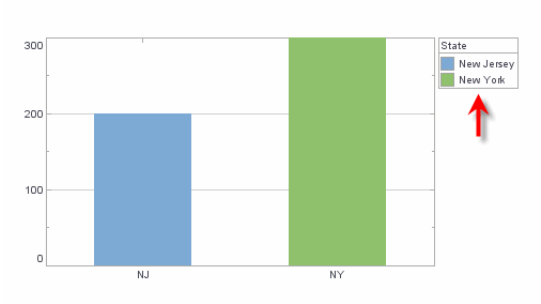
Parameter

frame a *TextFrame* object

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ",200], ["NY",300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var frame = new CategoricalColorFrame();
var tf = new DefaultTextFrame();
frame.setField("State");
tf.setText('NJ','New Jersey');
tf.setText('NY','New York');
var spec = new LegendSpec();
spec.setTextFrame(tf);
frame.setLegendSpec(spec);
elem.colorFrame = frame;
graph.addElement(elem);
```



See Also

DataSet, to use a Data Block or query as the *dataset* source.

LegendSpec.SETTEXTSPEC(SPEC)

Specifies the legend body text attributes, such as color, font, format, etc.

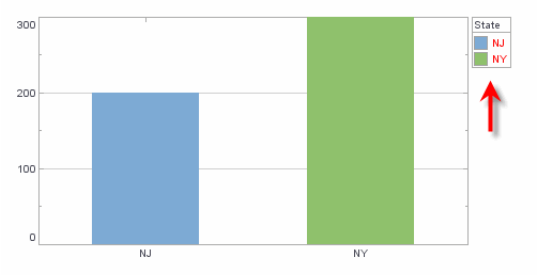
Parameter

spec a *TextSpec* object

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ",200], ["NY",300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var frame = new CategoricalColorFrame();
frame.setField("State");
var lspec = new LegendSpec();
var tspec = new TextSpec();
tspec.setColor(java.awt.Color(0xff0000))
lspec.setTextSpec(tspec);
frame.setLegendSpec(lspec);
elem.setColorFrame(frame);
graph.addElement(elem);
```



See Also

DataSet, to use a Data Block or query as the *dataset* source.

LegendSpec.SETTITLE(VALUE)

Specifies the legend title.

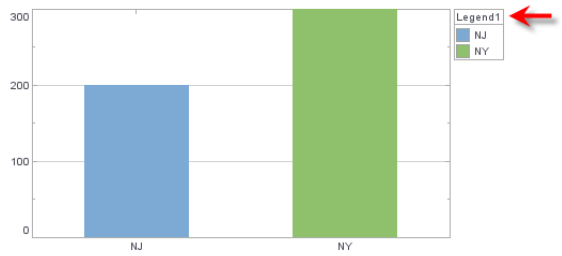
Parameter

value a String containing the title text

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ",200], ["NY",300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var frame = new CategoricalColorFrame();
frame.setField("State");
var spec = new LegendSpec();
spec.setTitle('Legend1');
frame.setLegendSpec(spec);
elem.setColorFrame(frame);
graph.addElement(elem);
```



See Also

DataSet, to use a Data Block or query as the *dataset* source.

LegendSpec.SETTITLETEXTSPEC(SPEC)

Specifies the legend title text attributes, such as color, font, format, etc.

Parameter

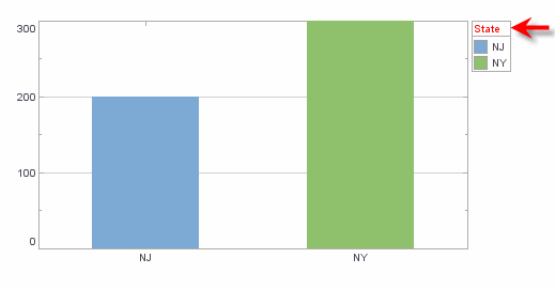
spec a *TextSpec* object

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ",200], ["NY",300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var frame = new CategoricalColorFrame();
frame.setField("State");
```

```
var lspec = new LegendSpec();
var tspec = new TextSpec();
tspec.setColor(java.awt.Color(0xff0000))
lspec.setTitleTextSpec(tspec);
frame.setLegendSpec(lspec);
elem.setColorFrame(frame);
graph.addElement(elem);
```



See Also

DataSet, to use a Data Block or query as the *dataset* source.

LegendSpec.SETTITLEVISIBLE(BOOLEAN)

Specifies whether the legend title is visible.

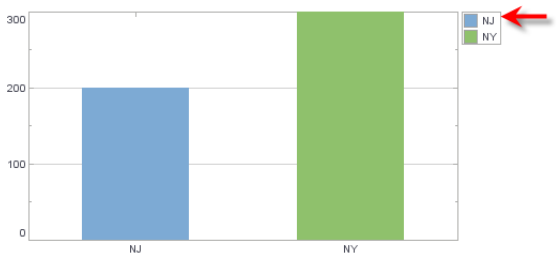
Parameter

boolean	true: show title false: hide title

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ",200], ["NY",300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var frame = new CategoricalColorFrame();
frame.setField("State");
var spec = new LegendSpec();
spec.setTitleVisible(false);
frame.setLegendSpec(spec);
elem.setColorFrame(frame);
graph.addElement(elem);
```



See Also

DataSet, to use a Data Block or query as the *dataset* source.

LegendSpec.SETVISIBLE(BOOLEAN)

Specifies whether the legend is visible.

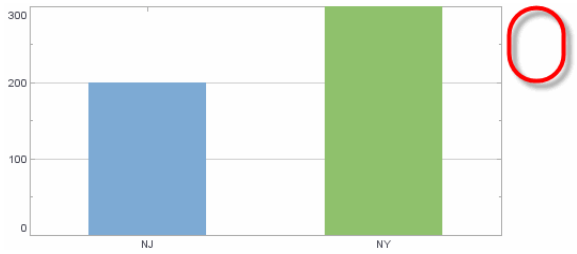
Type

boolean	true: show legend false: hide legend

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ",200], ["NY",300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var frame = new CategoricalColorFrame();
frame.setField("State");
var spec = new LegendSpec();
spec.setVisible(false);
frame.setLegendSpec(spec);
elem.setColorFrame(frame);
graph.addElement(elem);
```



See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.4.6 TitleSpec

The TitleSpec object contains title text and formatting information.

Full Name

```
inetsoft.graph.TitleSpec
```

TitleSpec: setLabel, setTextSpec

TitleSpec.SETLABEL(VALUE)

Specifies the title text.

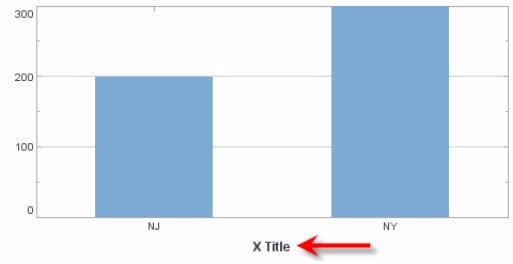
Parameter

value	a String containing the label

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ", 200], ["NY", 300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var spec = new TitleSpec();
spec.setLabel("X Title")
graph.setXTitleSpec(spec);
graph.addElement(elem);
```



See Also

DataSet, to use a Data Block or query as the *dataset* source.

TitleSpec.SETTEXTSPEC(SPEC)

Specifies the title text attributes, such as color, font, format, etc.

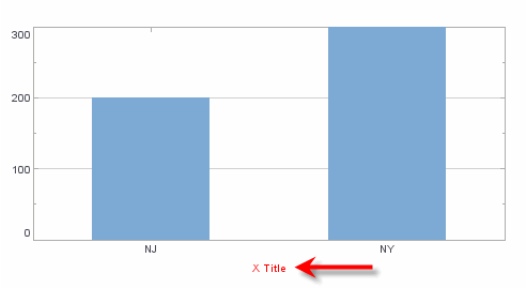
Parameter

spec a *TextSpec* object

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ", 200], ["NY", 300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var titlespec = new TitleSpec();
var textspec = new TextSpec();
textspec.setColor(java.awt.Color(0xff0000));
titlespec.setLabel("X Title")
titlespec.setTextSpec(textspec);
graph.setXTitleSpec(titlespec);
graph.addElement(elem);
```



See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.4.7 TextSpec

The TextSpec object contains information about the display of text.

Full Name

`inetsoft.graph.TextSpec`

TextSpec: setBackground, setColor, setFont, setFormat, setRotation

TextSpec.SETBACKGROUND(VALUE)

Specifies the label background color.

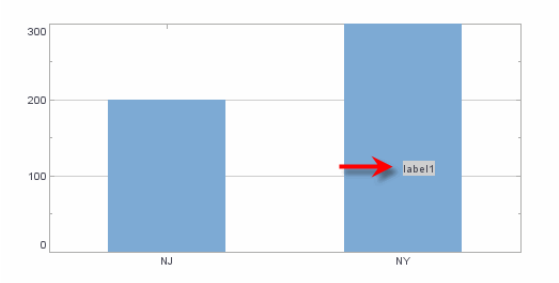
Parameter

value a `java.awt.Color` object

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ", 200], ["NY", 300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var form = new LabelForm();
form.setLabel("label1");
form.setValues(['NY', 100]);
var spec = new TextSpec();
spec.setBackground(java.awt.Color(0xcccccc));
form.setTextSpec(spec)
graph.addForm(form);
graph.addElement(elem);
```



See Also

DataSet, to use a Data Block or query as the *dataset* source.

TextSpec.SETCOLOR(VALUE)

Specifies the text color.

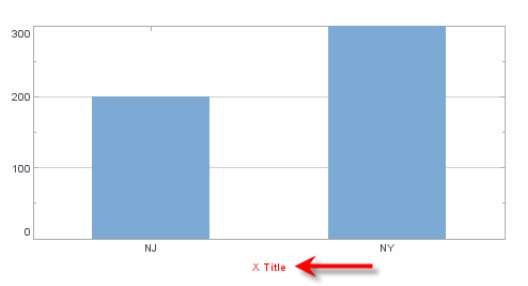
Parameter

value a `java.awt.Color` object

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ", 200], ["NY", 300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var titlespec = new TitleSpec();
var textspec = new TextSpec();
textspec.setColor(java.awt.Color(0xff0000));
titlespec.setLabel("X Title")
titlespec.setTextSpec(textspec);
graph.setXTitleSpec(titlespec);
graph.addElement(elem);
```



See Also

DataSet, to use a Data Block or query as the *dataset* source.

TextSpec.SETFONT(VALUE)

Specifies the text font, as a string with the following form: `'FontFamily-FontStyle-FontSize'`.

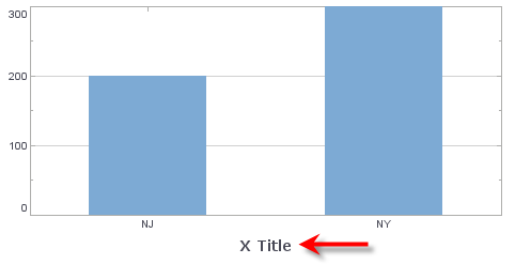
Parameter

value a java.awt.Font object

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ", 200], ["NY", 300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var titlespec = new TitleSpec();
var textspec = new TextSpec();
textspec.setFont(java.awt.Font('Verdana',java.awt.Font.BOLD, 14));
titlespec.setLabel("X Title")
titlespec.setTextSpec(textspec);
graph.setXTitleSpec(titlespec);
graph.addElement(elem);
```



The following values are available for font family, style, and size:

- `FontFamily` can be one of the available server fonts, as well as a generic family such as `serif` and `sans serif`.
- `FontStyle` can be `BOLD`, `ITALIC`, `PLAIN`, and `BOLD ITALIC`.
- `FontSize` can be any integer value.

See Also

DataSet, to use a Data Block or query as the *dataset* source.

TextSpec.SETFORMAT(FORMAT)

Specifies how date or numeric data should be represented in string format for display.

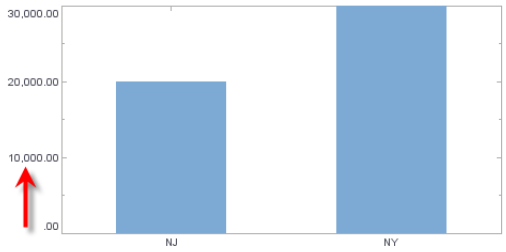
Parameter

format a java.text.Format object

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State","Quantity"],["NJ",20000],["NY",30000]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var qscale = new LinearScale("Quantity");
var aspec = new AxisSpec();
var tspec = new TextSpec();
tspec.setFormat(java.text.DecimalFormat("##,###.00"));
aspec.setTextSpec(tspec);
qscale.setAxisSpec(aspec);
graph.setScale("Quantity", qscale);
graph.addElement(elem);
```



Date masks follow the *java.text.SimpleDateFormat* format.

Date Format: For date formatting, use the following date masks:

```
M = Month
d = date
y = year
E = day of the week
```

Example:

```
For the date Nov 8, 2006:

M = 11; MM = 11; MMM = Nov; MMMM = November
d = 8; dd = 08;
yy = 06; yyyy = 2006
EEE = Wed
EEEE = Wednesday

Therefore, mask 'MMM-dd-yyyy' yields “Nov-08-2006.”
```

Numeric masks follow the *java.text.DecimalFormat* format.

Number Format: For number formatting, use the following numeric masks:

```
# = number
0 = Number with zero padding
```

Example:

```
For the number 124521.63:

#,###.##  yields 124,521.63
#,###.000 yields 124,521.630
```

Text Format: For text formatting, use `{0}` as a placeholder for the string data.

Example:

```
For the string 'Susan':

“Salesperson: {0}”  yields: “Salesperson: Susan”
“--{0}--” yields: “--Susan--”
```

See Also

DataSet, to use a Data Block or query as the *dataset* source.

TextSpec.SETROTATION(VALUE)

Specifies the text rotation in degrees.

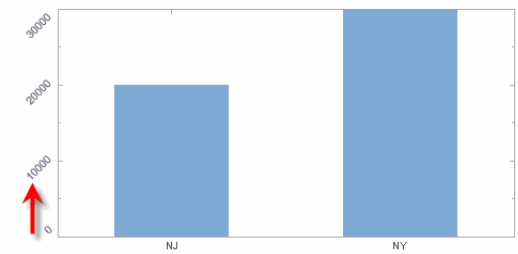
Parameter

value	a number of degrees

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State","Quantity"],["NJ",20000],["NY",30000]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var qscale = new LinearScale("Quantity");
var aspec = new AxisSpec();
var tspec = new TextSpec();
tspec.setRotation(45);
aspec.setTextSpec(tspec);
qscale.setAxisSpec(aspec);
graph.setScale("Quantity", qscale);
graph.addElement(elem);
```



See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.4.8 AxisSpec

The AxisSpec object holds axis information for a scale.

Full Name

```
inetsoft.graph.AxisSpec
```

AxisSpec: setAbbreviate, setAxisStyle, setGridAsShape, setGridColor, setGridOnTop, setGridStyle, setInPlot, setLabelVisible, setLineColor, setLineVisible, setTextFrame, setTextSpec, setTickVisible

AxisSpec.SETABBREVIATE(BOOLEAN)

Specifies that the common prefix of TimeScale labels should be omitted to conserve space. This property is only observed when a Date format is specified.

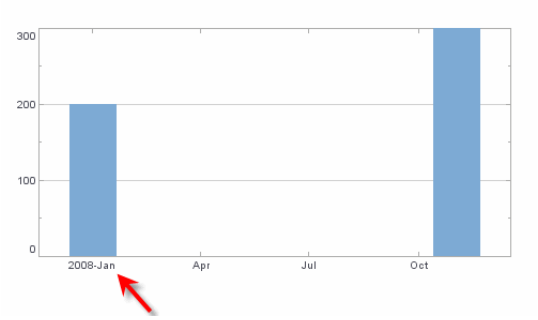
Parameter

boolean	true: drop common prefixes false: do not drop prefixes

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
var date1 = new Date();
var date2 = new Date();
date1.setFullYear(2008,0,1);
date2.setFullYear(2008,10,1);
dataset = [["Date", "Quantity"], [date1,200], [date2,300]];
graph = new EGraph();
var elem = new IntervalElement("Date", "Quantity");
var tscale = new TimeScale("Date");
var aspec = new AxisSpec();
var tspec = new TextSpec();
tspec.setFormat(java.text.SimpleDateFormat('yyyy-MMM'));
aspec.setTextSpec(tspec);
aspec.setAbbreviate(true);
tscale.setAxisSpec(aspec);
graph.setScale("Date", tscale);
graph.addElement(elem);
```



See Also

DataSet, to use a Data Block or query as the *dataset* source.

AxisSpec.SETALLTICKS(BOOLEAN)

Specifies whether tick marks should be shown for unlabeled data points.

Parameter

boolean	true: show all ticks false: show ticks for labeled points

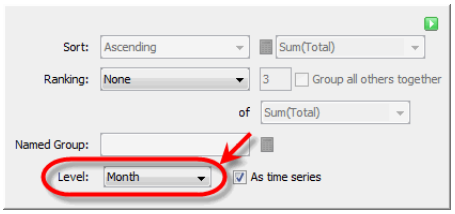
Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

For a report, add a blank chart and bind the chart to the sample 'All Sales' query. For a Viewsheet, create a new Viewsheet based on the sample 'All Sales' query, and add a blank chart. The follow the steps below:

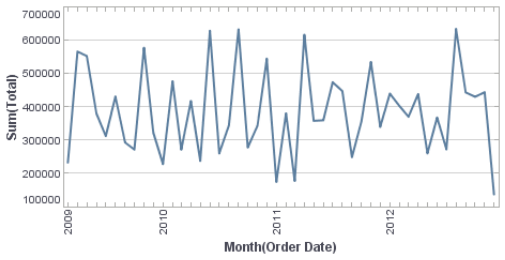
1. Open the Chart Editor for the chart, and drag 'Order Date' to the X-axis and 'Total' to the Y-axis.

2. Press the 'Edit Dimension' button next to the 'Order Date' field and select 'Month' from the 'Level' menu. Press the green 'Apply' button.



3. Add the following element-level script to the chart, and preview.

```
axis['Order Date'].format = [Chart.DATE_FORMAT,"yyyy"];
graph.getScale('Order
Date').getAxisSpec().setAllTicks(true);
```



Observe that ticks are displayed for every month, even though the 'yyyy' date format displays labels only for the years.

See Also

axis.ticksVisible, to set visibility of axis ticks.
LinearScale.setMinorIncrement(value), to set tick spacing for a linear scale.

AxisSpec.SETAXISSTYLE(VALUE)

Specifies the axis style.

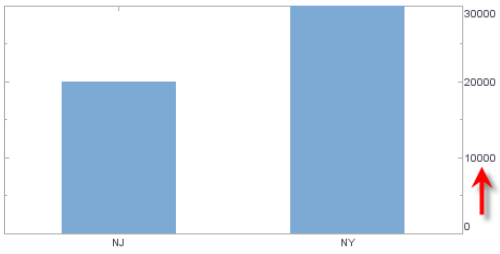
Parameter

value	AxisSpec.AXIS_SINGLE	(axes on left/bottom)
	AxisSpec.AXIS_SINGLE2	(axis on top/right)
	AxisSpec.AXIS_DOUBLE	(both axes, left/bot labels)
	AxisSpec.AXIS_DOUBLE2	(both axes, top/right labels)
	AxisSpec.AXIS_CROSS	(axis at zero-position)
	AxisSpec.AXIS_NONE	(axis not drawn)

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State","Quantity"],["NJ",20000],["NY",30000]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var qscale = new LinearScale("Quantity");
var aspec = new AxisSpec();
aspec.setAxisStyle(AxisSpec.AXIS_DOUBLE2);
qscale.setAxisSpec(aspec);
graph.setScale("Quantity", qscale);
graph.addElement(elem);
```



See Also

DataSet, to use a Data Block or query as the *dataset* source.

AxisSpec.SETGRIDASSHAPE(BOOLEAN)

Specifies whether axis grid lines are represented as shapes or positions. If represents as shapes, the grid lines can be transformed into curves under a coordinate transformation (for example, transformation of rectangular to polar coordinates). Otherwise, the grid lines remain straight, and only the endpoints are transformed.

Parameter

boolean	true if represented as shape false if represented as position

AxisSpec.SETGRIDCOLOR(VALUE)

Specifies the color of the axis grid lines.

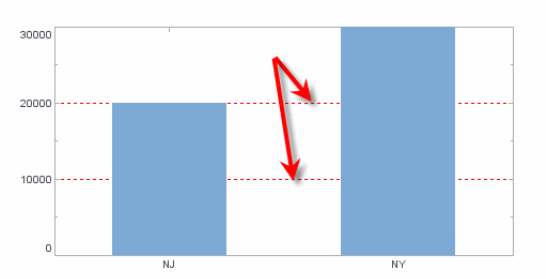
Parameter

value a `java.awt.Color` object

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State","Quantity"],["NJ",20000],["NY",30000]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var qscale = new LinearScale("Quantity");
var aspec = new AxisSpec();
aspec.setGridColor(java.awt.Color(0xff0000));
aspec.setGridStyle(Chart.DASH_LINE);
qscale.setAxisSpec(aspec);
graph.setScale("Quantity", qscale);
graph.addElement(elem);
```



See Also

DataSet, to use a Data Block or query as the *dataset* source.

AxisSpec.SETGRIDONTOP(BOOLEAN)

Specifies whether axis grid lines are layered over or under the chart elements.

Parameter

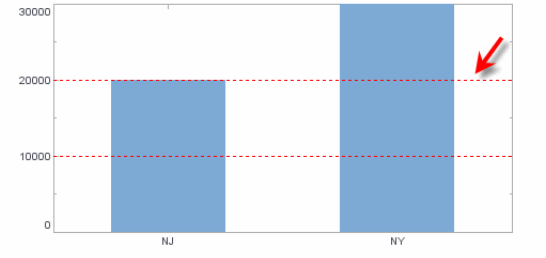
boolean true: grid overlays elements
false: elements overlay grid

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State","Quantity"],["NJ",20000],["NY",30000]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
```

```
var qscale = new LinearScale("Quantity");
var aspec = new AxisSpec();
aspec.setGridOnTop(true);
aspec.setGridColor(java.awt.Color(0xff0000));
aspec.setGridStyle(Chart.DASH_LINE);
qscale.setAxisSpec(aspec);
graph.setScale("Quantity", qscale);
graph.addElement(elem);
```



See Also

DataSet, to use a Data Block or query as the *dataset* source.

AxisSpec.SETGRIDSTYLE(VALUE)

Specifies the style of the axis grid lines.

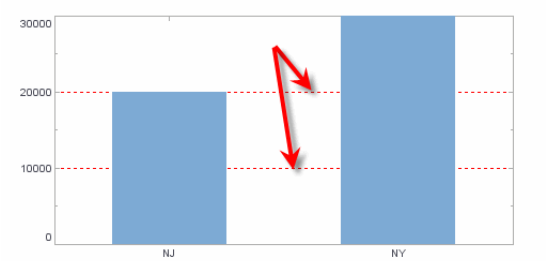
Parameter

value a *GLine* value

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State","Quantity"],["NJ",20000],["NY",30000]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var qscale = new LinearScale("Quantity");
var aspec = new AxisSpec();
aspec.setGridColor(java.awt.Color(0xff0000));
aspec.setGridStyle(Chart.DASH_LINE);
qscale.setAxisSpec(aspec);
graph.setScale("Quantity", qscale);
graph.addElement(elem);
```



See Also

DataSet, to use a Data Block or query as the *dataset* source.

AxisSpec.SETINPLOT(BOOLEAN)

Specifies whether the specified maximum measure value is included within the plot region. When `true`, this adds a slight buffer to the edge of the plot area in order to ensure that the specified maximum value falls within the displayed plot area.

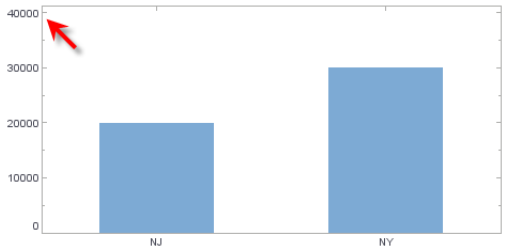
Parameter

boolean	`true: display max value within plot region` `false: display max value at edge of plot`

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State","Quantity"],["NJ",20000],["NY",30000]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var qscale = new LinearScale("Quantity");
qscale.setMax(40000);
var aspec = new AxisSpec();
aspec.setInPlot(true);
qscale.setAxisSpec(aspec);
graph.setScale("Quantity", qscale);
graph.addElement(elem);
```



See Also

GraphElement.setInPlot(boolean), to adjust plot boundaries to include graph elements.

GraphForm.setInPlot(boolean), to adjust plot boundaries to include graph forms.

LinearScale.setMax(value), for information on how to manually set a maximum value using the API method.

axis.maximum, for information on how to manually set a maximum value using the axis property.

DataSet, to use a Data Block or query as the *dataset* source.

AxisSpec.SETLABELVISIBLE(BOOLEAN)

Specifies whether the axis labels are visible or hidden.

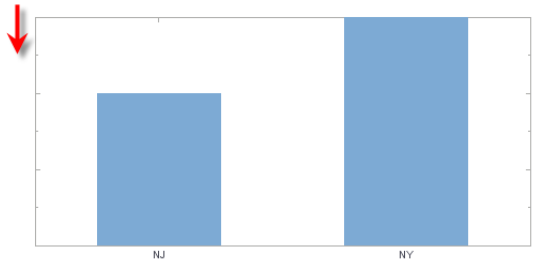
Parameter

boolean	true if visible false if not visible

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State","Quantity"],["NJ",20000],["NY",30000]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var qscale = new LinearScale("Quantity");
var aspec = new AxisSpec();
aspec.setLabelVisible(false);
qscale.setAxisSpec(aspec);
graph.setScale("Quantity", qscale);
graph.addElement(elem);
```



See Also

DataSet, to use a Data Block or query as the *dataset* source.

AxisSpec.SETLINECOLOR(VALUE)

Specifies the color of the axis lines.

Parameter

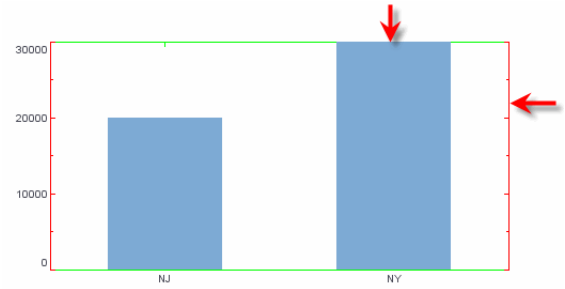
value	a java.awt.Color object

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State","Quantity"],["NJ",20000],["NY",30000]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var qscale = new LinearScale("Quantity");
```

```
var cscale = new CategoricalScale("State");
var aspec1 = new AxisSpec();
var aspec2 = new AxisSpec();
aspec1.setLineColor(java.awt.Color(0xff0000));
aspec2.setLineColor(java.awt.Color(0x00ff00));
qscale.setAxisSpec(aspec1);
cscale.setAxisSpec(aspec2);
graph.setScale("Quantity", qscale);
graph.setScale("State", cscale);
graph.addElement(elem);
```



See Also

DataSet, to use a Data Block or query as the *dataset* source.

AxisSpec.SETLINEVISIBLE(BOOLEAN)

Specifies whether the axis lines are visible or hidden.

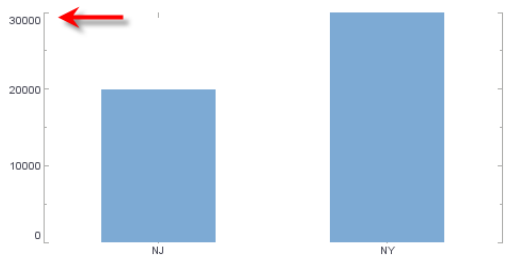
Parameter

boolean	true if visible false if not visible

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State","Quantity"],["NJ",20000],["NY",30000]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var qscale = new LinearScale("Quantity");
var cscale = new CategoricalScale("State");
var aspec1 = new AxisSpec();
var aspec2 = new AxisSpec();
aspec1.setLineVisible(true);
aspec2.setLineVisible(false);
qscale.setAxisSpec(aspec1);
cscale.setAxisSpec(aspec2);
graph.setScale("Quantity", qscale);
graph.setScale("State", cscale);
graph.addElement(elem);
```



See Also

DataSet, to use a Data Block or query as the *dataset* source.

AxisSpec.SETTEXTFRAME(FRAME)

Specifies a mapping between axis values and replacement text.

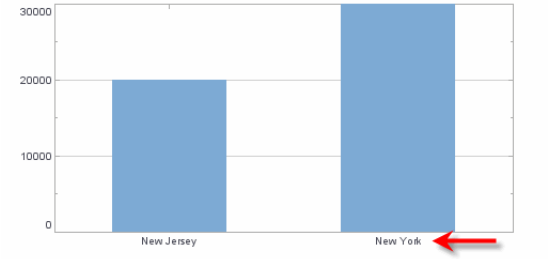
Parameter

frame a *TextFrame* object

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State","Quantity"],["NJ",20000],["NY",30000]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var cscale = new CategoricalScale("State");
var tframe = new DefaultTextFrame();
tframe.setText('NJ','New Jersey');
tframe.setText('NY','New York');
var aspec = new AxisSpec();
aspec.setTextFrame(tframe);
cscale.setAxisSpec(aspec);
graph.setScale("State", cscale);
graph.addElement(elem);
```



See Also

DataSet, to use a Data Block or query as the *dataset* source.

AxisSpec.SETTEXTSPEC(SPEC)

Specifies the display of axis text.

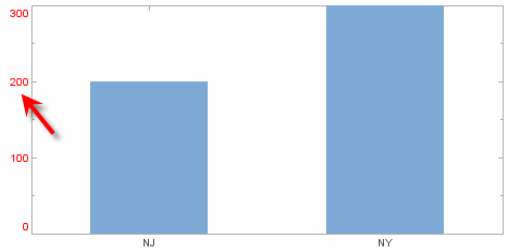
Parameter

spec	a *TextSpec* object

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ", 200], ["NY", 300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var qscale = new LinearScale("Quantity");
var aspec = new AxisSpec();
var textspec = new TextSpec();
textspec.setColor(java.awt.Color(0xff0000));
aspec.setTextSpec(textspec);
qscale.setAxisSpec(aspec);
graph.setScale("Quantity", qscale);
graph.addElement(elem);
```



See Also

DataSet, to use a Data Block or query as the *dataset* source.

AxisSpec.SETTICKVISIBLE(BOOLEAN)

Specifies whether the axis tick marks are visible or hidden.

Parameter

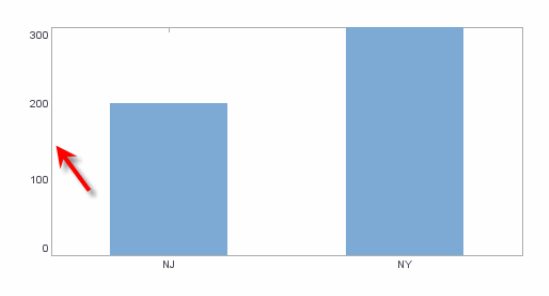
boolean	true if visible false if not visible

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ", 200], ["NY", 300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var qscale = new LinearScale("Quantity");
```

```
var aspec = new AxisSpec();
aspec.setTickVisible(false);
qscale.setAxisSpec(aspec);
graph.setScale("Quantity", qscale);
graph.addElement(elem);
```



See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.4.9 PlotSpec

The PlotSpec object contains visual information for a *Coordinate* object (i.e., the Chart's plot area).

Full Name

`inetsoft.graph.PlotSpec`

PlotSpec: alpha, setBackground, setBackgroundImage, setLockAspect, setXMax, setXMin, setYMax, setYMin

PlotSpec.SETALPHA(VALUE)

Specifies the transparency level for the *Coordinate* object background color or background image.

Parameter

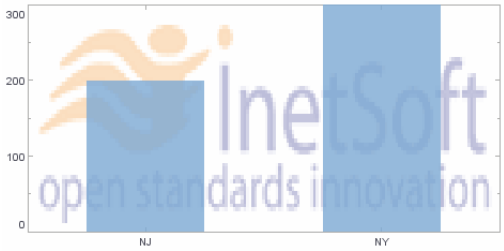
value	a number in range [0,1], where: 0 = completely transparent 1 = completely opaque

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State","Quantity"], ["NJ",200], ["NY",300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
elem.setHint(GraphElement.HINT_ALPHA,.8)
var sscale = new CategoricalScale("State");
var qscale = new LinearScale("Quantity");
var coord = new RectCoord(sscale,qscale);
var spec = new PlotSpec();
```

```
var logo = getImage("http://www.inetsoft.com/images/home/
logo.gif");
spec.setBackgroundImage(logo);
spec.setAlpha(.3);
coord.setPlotSpec(spec);
graph.setCoordinate(coord);
graph.addElement(elem);
```



See Also

DataSet, to use a Data Block or query as the *dataset* source.

PlotSpec.SETBACKGROUND(VALUE)

Specifies the background color for a *Coordinate* object (i.e., the plot area).

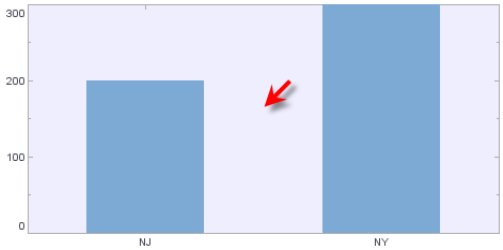
Parameter

value a `java.awt.Color` object

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ", 200], ["NY", 300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var sscale = new CategoricalScale("State");
var qscale = new LinearScale("Quantity");
var coord = new RectCoord(sscale,qscale);
var spec = new PlotSpec();
spec.setBackground(java.awt.Color(0xEEEEFF));
coord.setPlotSpec(spec);
graph.setCoordinate(coord);
graph.addElement(elem);
```



See Also

DataSet, to use a Data Block or query as the *dataset* source.

PlotSpec.SETBACKGROUNDIMAGE(VALUE)

Specifies the background image for a *Coordinate* object (i.e., the plot area).

Parameter

`Image object, see getImage()`

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State","Quantity"], ["NJ",200], ["NY",300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
elem.setHint(GraphElement.HINT_ALPHA,.8)
var sscale = new CategoricalScale("State");
var qscale = new LinearScale("Quantity");
var coord = new RectCoord(sscale,qscale);
var spec = new PlotSpec();
var logo = getImage("http://www.inetsoft.com/images/home/
logo.gif");
spec.setBackgroundImage(logo);
coord.setPlotSpec(spec);
graph.setCoordinate(coord);
graph.addElement(elem);
```



See Also

DataSet, to use a Data Block or query as the *dataset* source.

PlotSpec.SETLOCKASPECT(BOOLEAN)

Specifies that the original image aspect ratio should be retained, with coordinate scaling adapted to the image. If set to false (default), the image is resized to fit the existing coordinate scaling.

Parameter

boolean	true: keep original aspect ratio false: change ratio to fit plot area (default)

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and

Adding Component Script in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State","Quantity"], ["NJ",200], ["NY",300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
elem.setHint(GraphElement.HINT_ALPHA,.8)
var sscale = new CategoricalScale("State");
var qscale = new LinearScale("Quantity");
var coord = new RectCoord(sscale,qscale);
var spec = new PlotSpec();
spec.setLockAspect(true);
var logo = getImage("http://www.inetsoft.com/images/home/
logo.gif");
spec.setBackgroundImage(logo);
coord.setPlotSpec(spec);
graph.setCoordinate(coord);
graph.addElement(elem);
```

Note: The gray background is set from the 'Chart Properties' dialog box (Style Studio) or the 'Format' dialog box (Viewsheet).



See Also

DataSet, to use a Data Block or query as the *dataset* source.

PlotSpec.SETXMAX(VALUE)

Specifies the X-axis value at which to place the right edge of the background image.

Parameter

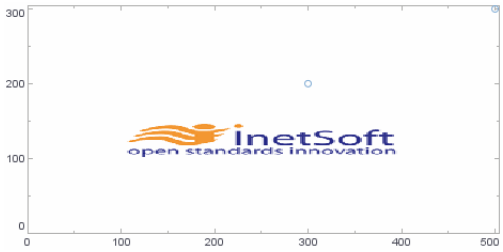
value a number

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["Q1","Q2"], [300,200], [500,300]];
graph = new EGraph();
var elem = new PointElement("Q1", "Q2");
elem.setHint(GraphElement.HINT_ALPHA,.8)
var sscale = new LinearScale("Q1");
var qscale = new LinearScale("Q2");
var coord = new RectCoord(sscale,qscale);
var spec = new PlotSpec();
var logo =
getImage("http://www.inetsoft.com/images/home/logo.gif");
spec.setBackgroundImage(logo);
spec.setYMax(150)
spec.setYMin(100)
spec.setXMax(400)
```

```
spec.setXMin(100)
coord.setPlotSpec(spec);
graph.setCoordinate(coord);
graph.addElement(elem);
```



See Also

DataSet, to use a Data Block or query as the *dataset* source.

PlotSpec.SETXMIN(VALUE)

Specifies the X-axis value at which to place the left edge of the background image.

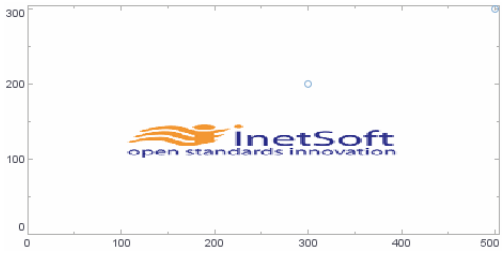
Parameter

value a number

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["Q1","Q2"], [300,200], [500,300]];
graph = new EGraph();
var elem = new PointElement("Q1", "Q2");
elem.setHint(GraphElement.HINT_ALPHA,.8)
var sscale = new LinearScale("Q1");
var qscale = new LinearScale("Q2");
var coord = new RectCoord(sscale,qscale);
var spec = new PlotSpec();
var logo =
getImage("http://www.inetsoft.com/images/home/logo.gif");
spec.setBackgroundImage(logo);
spec.setYMax(150)
spec.setYMin(100)
spec.setXMax(400)
spec.setXMin(100)
coord.setPlotSpec(spec);
graph.setCoordinate(coord);
graph.addElement(elem);
```



See Also

DataSet, to use a Data Block or query as the *dataset* source.

PlotSpec.SETYMAX(VALUE)

Specifies the Y-axis value at which to place the top edge of the background image.

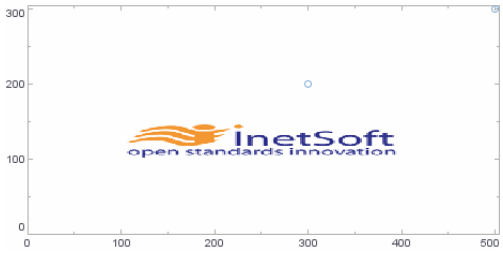
Parameter

value a number

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["Q1","Q2"], [300,200], [500,300]];
graph = new EGraph();
var elem = new PointElement("Q1", "Q2");
elem.setHint(GraphElement.HINT_ALPHA,.8)
var sscale = new LinearScale("Q1");
var qscale = new LinearScale("Q2");
var coord = new RectCoord(sscale,qscale);
var spec = new PlotSpec();
var logo =
getImage("http://www.inetsoft.com/images/home/logo.gif");
spec.setBackgroundImage(logo);
spec.setYMax(150)
spec.setYMin(100)
spec.setXMax(400)
spec.setXMin(100)
coord.setPlotSpec(spec);
graph.setCoordinate(coord);
graph.addElement(elem);
```



See Also

DataSet, to use a Data Block or query as the *dataset* source.

PlotSpec.SETYMIN(VALUE)

Specifies the Y-axis value at which to place the bottom edge of the background image.

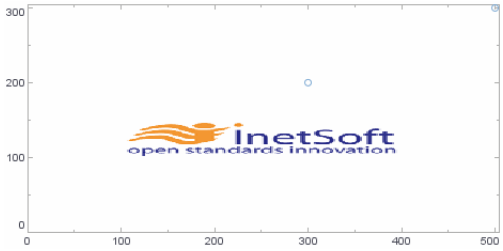
Parameter

value a number

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["Q1","Q2"], [300,200], [500,300]];
graph = new EGraph();
var elem = new PointElement("Q1", "Q2");
elem.setHint(GraphElement.HINT_ALPHA,.8)
var sscale = new LinearScale("Q1");
var qscale = new LinearScale("Q2");
var coord = new RectCoord(sscale,qscale);
var spec = new PlotSpec();
var logo =
getImage("http://www.inetsoft.com/images/home/logo.gif");
spec.setBackgroundImage(logo);
spec.setYMax(150)
spec.setYMin(100)
spec.setXMax(400)
spec.setXMin(100)
coord.setPlotSpec(spec);
graph.setCoordinate(coord);
graph.addElement(elem);
```



See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.5 Chart Elements

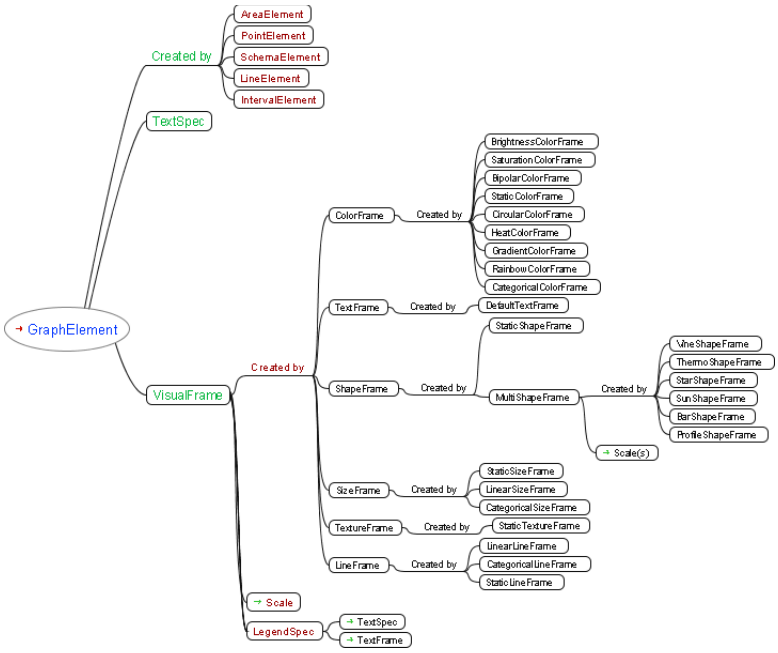
This section discusses the various data-representation elements that can be added to charts.

CR.5.1 GraphElement

The GraphElement object contains the visual elements that represent data. For example, PointElement is a GraphElement that represents data tuples as points.

Full Name

`inetsoft.graph.element.GraphElement`



GraphElement: addDim(), addVar(), setCollisionModifier, setColorFrame, setInPlot, setLabelPlacement, setLineFrame, setHint(), setShapeFrame, setSizeFrame, setTextFrame, setTextSpec, setTextureFrame

Created by

AreaElement
PointElement
SchemaElement
LineElement
IntervalElement

GraphElement.ADDDIM(FIELD)

Add a dimension to a GraphElement object. A dimension is plotted on the X-axis, or on the outer coordinates of nested coordinates.

Parameter

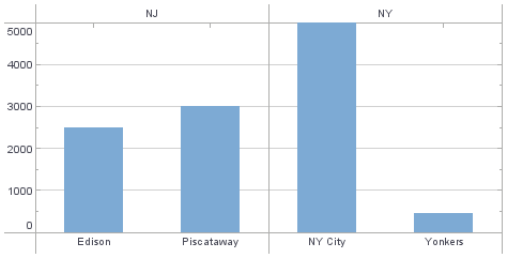
`field` `String containing name of dimension`

`Note: Script that modifies 'graph' should be placed at the element level. See` *`Adding Element-Level Script`* `in` *`Report Scripting`* `and`

Adding Component Script in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "City", "Quantity"],
           ["NJ","Edison",2500], ["NJ","Piscataway",3000],
           ["NY","NY City",5000],["NY","Yonkers",450]];
graph = new EGraph();
var elem = new IntervalElement("State","Quantity");
elem.addDim("City");
graph.addElement(elem);
```



See Also

DataSet, to use a Data Block or query as the *dataset* source.

GraphElement.ADDVAR(FIELD)

Add a variable to a GraphElement object. A variable is plotted on the Y-axis.

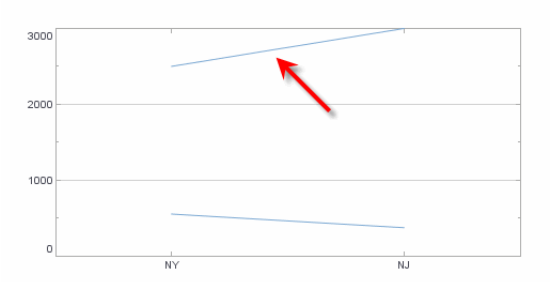
Parameter

field String containing name of variable

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity",
"Total"],["NY",550,2500],["NJ",370,3000]];
graph = new EGraph();
var elem = new LineElement("State","Quantity");
elem.addVar("Total");
graph.addElement(elem);
```



See Also

DataSet, to use a Data Block or query as the *dataset* source.

GraphElement.SETCOLLISIONMODIFIER(VALUE)

Specifies how collisions (elements occupying the same location) should be handled.

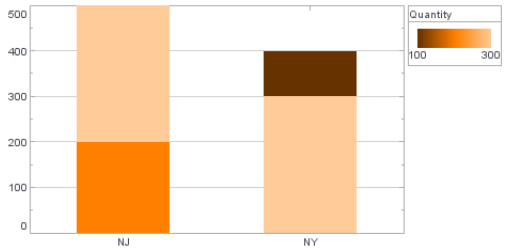
Parameter

value GraphElement.MOVE_NONE (do not stack/center)
GraphElement.MOVE_CENTER (center, do not stack)
GraphElement.MOVE_DODGE (offset horizontal)
GraphElement.MOVE_STACK (offset vertical (stack))
GraphElement.MOVE_JITTER (random offset (points))
GraphElement.DODGE_SYMMETRIC (offset horiz, center)
GraphElement.STACK_SYMMETRIC (offset vert, center)

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

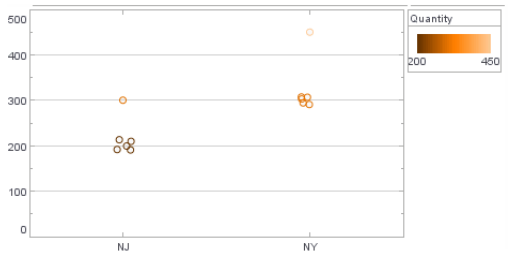
Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ", 200], ["NJ", 300],
["NY", 300], ["NY", 100]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var scale = new LinearScale("Quantity");
var frame = new HeatColorFrame();
frame.setField("Quantity");
elem.setColorFrame(frame);
elem.setStackGroup(true);
elem.setCollisionModifier(GraphElement.STACK_SYMMETRIC);
graph.addElement(elem);
```



Example (Report or Viewsheet)

```
dataset = [["State","Quantity"],["NJ",200],["NJ",200],
          ["NJ",200],["NJ",200],["NJ",200],["NJ",300],
          ["NY",300],["NY",300],["NY",300],["NY",300],
          ["NY",300],["NY",450]];
graph = new EGraph();
var elem = new PointElement("State", "Quantity");
var frame = new HeatColorFrame();
frame.setField("Quantity");
elem.setColorFrame(frame);
elem.setStackGroup(true);
elem.setCollisionModifier(GraphElement.MOVE_JITTER);
graph.addElement(elem);
```



See Also

DataSet, to use a Data Block or query as the *dataset* source.

GraphElement.SETCOLORFRAME(FRAME)

Specifies the aesthetic color treatment for the chart elements. A *ColorFrame* can color-code chart elements by value, or simply specify a static color scheme.

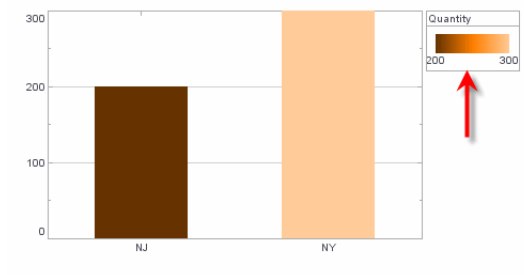
Parameter

frame a *ColorFrame* object

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ", 200], ["NY", 300]];
graph = new EGraph();
var frame = new HeatColorFrame();
var elem = new IntervalElement("State", "Quantity");
frame.setField("Quantity");
elem.setColorFrame(frame);
graph.addElement(elem);
```



See Also

DataSet, to use a Data Block or query as the *dataset* source.

GraphElement.SETINPLOT(BOOLEAN)

Specifies whether the chart should be resized so that graph elements remain fully visible in the chart area.

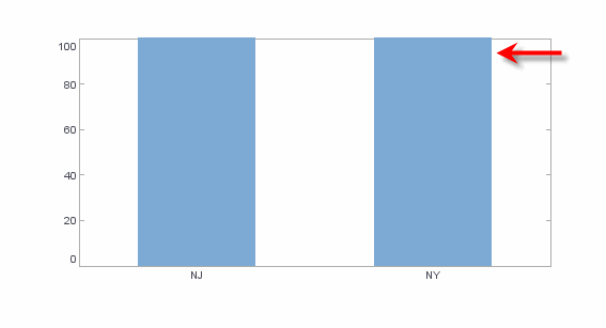
Parameter

boolean	true: resize chart false: do not resize chart (crop elements)

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ", 200], ["NY", 300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var scale = new LinearScale("Quantity");
scale.setMax(100);
graph.setScale("Quantity",scale);
elem.setInPlot(false);
graph.addElement(elem);
```



See Also

AxisSpec.setInPlot(boolean), to adjust plot boundaries to include graph labels.

GraphForm.setInPlot(boolean), to adjust plot boundaries to include graph forms.

DataSet, to use a Data Block or query as the *dataset* source.

GraphElement.SETLABELPLACEMENT(VALUE)

Specifies the location of element labels.

Parameter

value	Chart.CENTER (center of element) Chart.BOTTOM (below element) Chart.TOP (above element Chart.RIGHT (right of element) Chart.LEFT (left of element)

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["CA", 200], ["NY", 300]];
graph = new EGraph();
var elem = new PointElement("State", "Quantity");
var frame = new DefaultTextFrame();
frame.setField("Quantity")
elem.setTextFrame(frame);
elem.setLabelPlacement(Chart.BOTTOM);
graph.addElement(elem);
```



See Also

DataSet, to use a Data Block or query as the *dataset* source.

GraphElement.SETLINEFRAME(FRAME)

Specifies the aesthetic line style of graphical elements. A *LineFrame* can line-code chart elements by value, or simply specify a static line style.

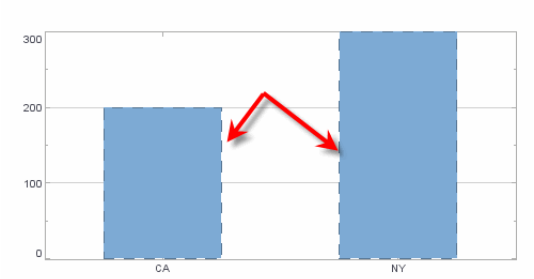
Parameter

frame a *LineFrame* object

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["CA", 200], ["NY", 300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var frame = new StaticLineFrame();
frame.setLine(GLine.LARGE_DASH)
elem.setLineFrame(frame);
graph.addElement(elem);
```



See Also

DataSet, to use a Data Block or query as the *dataset* source.

GraphElement.SETHINT(TYPE,VALUE)

Add an effect to a GraphElement object. The available effects are `HINT_EXPLODED` (element separation, or explosion), `HINT_SHINE` (three-dimensional shading), and `HINT_ALPHA` (transparency).

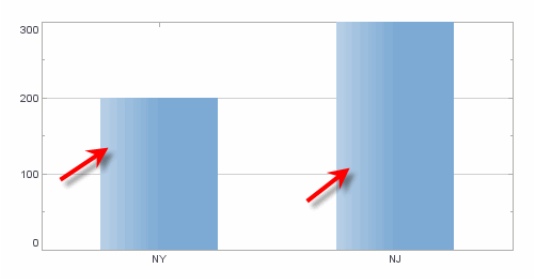
Parameters

type	GraphElement.HINT_EXPLODED, GraphElement.HINT_SHINE, or GraphElement.HINT_ALPHA
value	For HINT_EXPLODED: 'true' or 'false' For INT_SHINE: 'true' or 'false' For HINT_ALPHA: float in range [0,1]

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"],["NY",200], ["NJ",300]];
graph = new EGraph();
var elem = new IntervalElement("State","Quantity");
elem.setHint(GraphElement.HINT_SHINE,'true')
graph.addElement(elem);
```



See Also

DataSet, to use a Data Block or query as the *dataset* source.

GraphElement.SETSHAPEFRAME(FRAME)

Specifies the aesthetic shape treatment for the chart elements. A *ShapeFrame* can shape-code chart elements by value, or simply specify a static shape style.

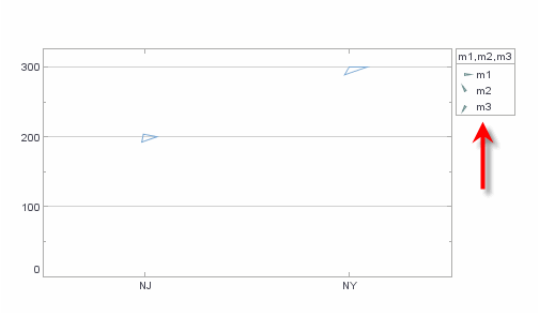
Parameter

frame	a *ShapeFrame* object

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State","Quantity","m1","m2",
"m3"],["NJ",200,6,3,4],["NY",300,8,2,5]];
graph = new EGraph();
var frame = new StarShapeFrame();
var elem = new PointElement("State", "Quantity");
frame.setFields(["m1", "m2", "m3"]);
elem.setShapeFrame(frame);
graph.addElement(elem);
```



See Also

DataSet, to use a Data Block or query as the *dataset* source.

GraphElement.SETSIZEFRAME(FRAME)

Specifies the size of graphical elements. A *SizeFrame* can size-code chart elements by value, or simply specify a static size.

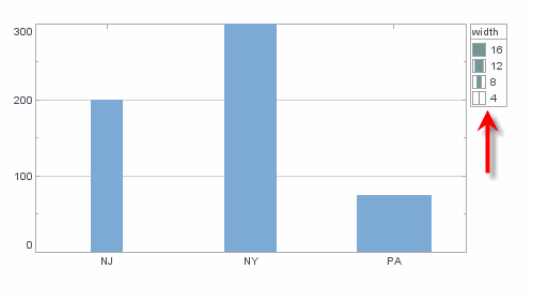
Parameter

frame a *SizeFrame* object

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset =
[["State","Quantity","width"],["NJ",200,5],["NY",300,10],["
PA",75,15]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity")
var frame = new LinearSizeFrame();
frame.setField("width");
frame.setSmallest(10);
frame.setLargest(50);
frame.setMax(100);
elem.setSizeFrame(frame);
graph.addElement(elem);
```



See Also

DataSet, to use a Data Block or query as the *dataset* source.

GraphElement.SETTEXTFRAME(FRAME)

Specifies the data values to be displayed on the chart elements as text, as well as the mapping between element values and displayed text.

Parameter

frame a *TextFrame* object

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["CA", 200], ["NY", 300]];
graph = new EGraph();
var elem = new PointElement("State", "Quantity");
var frame = new DefaultTextFrame();
frame.setField("Quantity");
elem.setTextFrame(frame);
graph.addElement(elem);
```



See Also

DataSet, to use a Data Block or query as the *dataset* source.

GraphElement.SETTEXTSPEC(SPEC)

Specifies the text attributes such as color, font, format, etc.

Parameter

spec a *TextSpec* object

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["CA", 200], ["NY", 300]];
graph = new EGraph();
var elem = new PointElement("State", "Quantity");
var frame = new DefaultTextFrame();
frame.setField("Quantity");
var spec = new TextSpec();
spec.setColor(java.awt.Color(0xff0000));
elem.setTextFrame(frame);
elem.setTextSpec(spec);
graph.addElement(elem);
```



See Also

DataSet, to use a Data Block or query as the *dataset* source.

GraphElement.SETTEXTUREFRAME(FRAME)

Specifies the aesthetic texture of graphical elements. A *TextureFrame* can texture-code chart elements by value, or simply specify a static texture style.

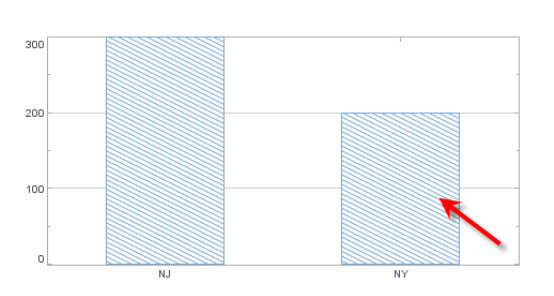
Parameter

frame a *TextureFrame* object

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ", 300], ["NY", 200]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var frame = new StaticTextureFrame();
frame.setTexture(GTexture.PATTERN_18)
elem.setTextureFrame(frame);
graph.addElement(elem);
```



See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.5.2 IntervalElement

The IntervalElement object contains the visual elements for bar charts and range visualization. To create a new IntervalElement object, call the IntervalElement constructor.

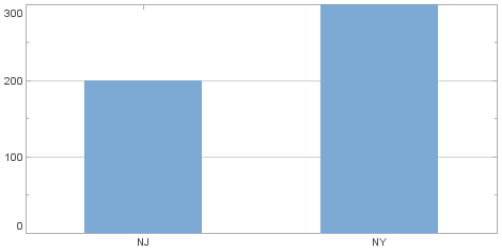
```
var elem = new IntervalElement("State", "Quantity");
```

You can pass a set of fields as input to the constructor (e.g., 'State', 'Quantity'), or specify these later using the `addDim()` and `addVar()` methods.

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ",200], ["NY",300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
graph.addElement(elem);
```



Full Name

```
inetsoft.graph.element.IntervalElement
```

IntervalElement: addInterval(), setStackGroup, setStackNegative

Inherits from

GraphElement: addDim(), addVar(), setCollisionModifier, setColorFrame, setInPlot, setLabelPlacement, setLineFrame, setHint(), setShapeFrame, setSizeFrame, setTextFrame, setTextSpec, setTextureFrame

See Also

DataSet, to use a Data Block or query as the *dataset* source.

IntervalElement.ADDINTERVAL(LOWER,UPPER)

Adds a "floating" interval element (bar) with lower and upper values specified by the inputs.

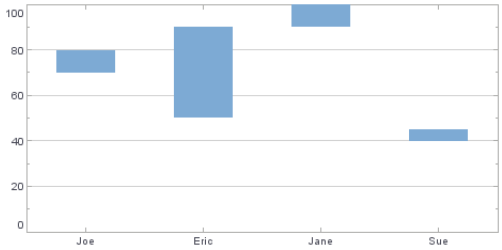
Parameters

lower	Name of field defining lower values (String)
upper	Name of field defining upper values (String)

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["Student","Bottom Score","Top
Score"],["Joe",70,80], ["Eric",50,90],["Jane",90,100],
["Sue",40,45]];
graph = new EGraph();
var elem = new IntervalElement();
elem.addDim("Student");
elem.addInterval("Bottom Score","Top Score")
graph.addElement(elem);
```



Inherits from

GraphElement: addDim(), addVar(), setCollisionModifier, setColorFrame, setInPlot, setLabelPlacement, setLineFrame, setHint(), setShapeFrame, setSizeFrame, setTextFrame, setTextSpec, setTextureFrame

See Also

DataSet, to use a Data Block or query as the *dataset* source.

IntervalElement.SETSTACKGROUP(BOOLEAN)

Specifies whether each element group (i.e., bar series) should be stacked independently, or whether all should form a single stack order.

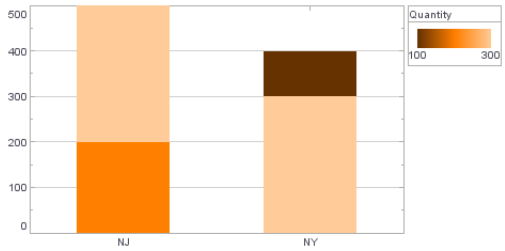
Parameters

boolean	true: independent stack order for each group false: single stack order for all groups

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State","Quantity"],["NJ",200],
["NJ",300],["NY",300],["NY",100]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var scale = new LinearScale("Quantity");
var frame = new HeatColorFrame();
frame.setField("Quantity");
elem.setColorFrame(frame);
elem.setStackGroup(true);
elem.setCollisionModifier(GraphElement.STACK_SYMMETRIC);
graph.addElement(elem);
```



Inherits from

GraphElement: addDim(), addVar(), setCollisionModifier, setColorFrame, setInPlot, setLabelPlacement, setLineFrame, setHint(), setShapeFrame, setSizeFrame, setTextFrame, setTextSpec, setTextureFrame

See Also

DataSet, to use a Data Block or query as the *dataset* source.

IntervalElement.SETSTACKNEGATIVE(BOOLEAN)

Specifies whether negative and positive values stack independently on opposite sides of the axis, or whether stacking is cumulative (i.e., determined arithmetically).

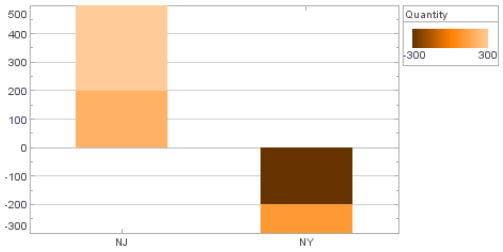
Parameters

boolean	true: independent stack order for each group false: single stack order for all groups

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset =
[["State","Quantity"],["NJ",200],["NJ",300],["NY",-
300],["NY",100]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var scale = new LinearScale("Quantity");
var frame = new HeatColorFrame();
frame.setField("Quantity");
elem.setColorFrame(frame);
elem.setStackGroup(true);
elem.setStackNegative(false);
elem.setCollisionModifier(GraphElement.STACK_SYMMETRIC);
graph.addElement(elem);
```



Inherits from

GraphElement: addDim(), addVar(), setCollisionModifier, setColorFrame, setInPlot, setLabelPlacement, setLineFrame, setHint(), setShapeFrame, setSizeFrame, setTextFrame, setTextSpec, setTextureFrame

See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.5.3 LineElement

The LineElement object contains the visual elements for a line chart. To create a new LineElement object, call the LineElement constructor.

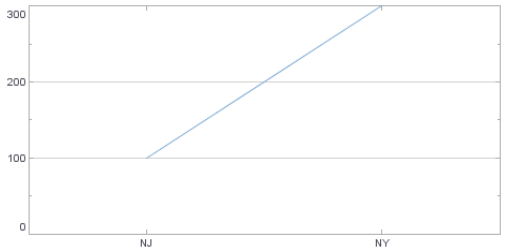
```
var elem = new LineElement("State", "Quantity");
```

You can pass one, two, or three field names (e.g., 'State', 'Quantity') to the constructor, or specify these later using the `addDim()` and `addVar()` methods.

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ", 100], ["NY", 300]];
graph = new EGraph();
var elem = new LineElement("State", "Quantity")
graph.addElement(elem);
```



Full Name

`inetsoft.graph.element.LineElement`

LineElement: setClosed, setEndArrow, setStackGroup, setStackNegative, setStartArrow

Inherits from

GraphElement: addDim(), addVar(), setCollisionModifier, setColorFrame, setInPlot, setLabelPlacement, setLineFrame, setHint(), setShapeFrame, setSizeFrame, setTextFrame, setTextSpec, setTextureFrame

See Also

DataSet, to use a Data Block or query as the *dataset* source.

LineElement.SETCLOSED(BOOLEAN)

Specifies whether the line object should be automatically closed (i.e., endpoints connected).

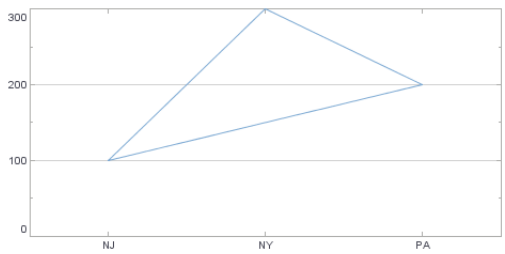
Type

boolean true: close the figure
false: do not close

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset =
[["State","Quantity"],["NJ",100],["NY",300],["PA",200]];
graph = new EGraph();
elem = new LineElement("State", "Quantity")
elem.setClosed(true);
graph.addElement(elem);
```



Inherits from

GraphElement: addDim(), addVar(), setCollisionModifier, setColorFrame, setInPlot, setLabelPlacement, setLineFrame, setHint(), setShapeFrame, setSizeFrame, setTextFrame, setTextSpec, setTextureFrame

See Also

DataSet, to use a Data Block or query as the *dataset* source.

LineElement.SETENDARROW(BOOLEAN)

Specifies whether an arrow should be drawn at the end of the line (last point).

Parameters

boolean	true: draw arrow false: do not draw arrow

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset =
[["State","Quantity"],["NJ",100],["NY",300],["PA",200]];
graph = new EGraph();
elem = new LineElement("State", "Quantity")
elem.setEndArrow(true);
graph.addElement(elem);
```

Inherits from

GraphElement: addDim(), addVar(), setCollisionModifier, setColorFrame, setInPlot, setLabelPlacement, setLineFrame, setHint(), setShapeFrame, setSizeFrame, setTextFrame, setTextSpec, setTextureFrame

See Also

DataSet, to use a Data Block or query as the *dataset* source.

LineElement.SETSTACKGROUP(BOOLEAN)

Specifies whether each subgroup receives an independent element (separate line), or whether a single element is used for all.

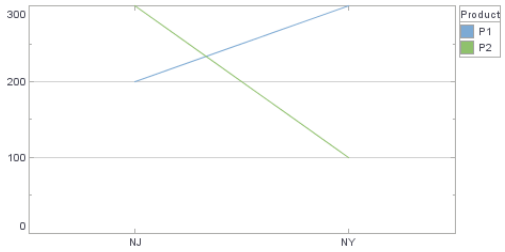
Parameters

boolean	true: independent lines for each group false: single line for all groups (default)

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset =
[["State","Product","Quantity"],["NJ","P1",200],["NJ","P2",
300], ["NY","P1",300],["NY","P2",100]];
graph = new EGraph();
var elem = new LineElement("State", "Quantity");
elem.setColorFrame(new CategoricalColorFrame("Product"));
elem.setStackGroup(true);
graph.addElement(elem);
```



Inherits from

GraphElement: addDim(), addVar(), setCollisionModifier, setColorFrame, setInPlot, setLabelPlacement, setLineFrame, setHint(), setShapeFrame, setSizeFrame, setTextFrame, setTextSpec, setTextureFrame

See Also

DataSet, to use a Data Block or query as the *dataset* source.

LineElement.SETSTACKNEGATIVE(BOOLEAN)

Specifies whether negative and positive values stack independently on opposite sides of the axis, or whether stacking is cumulative (i.e., determined arithmetically).

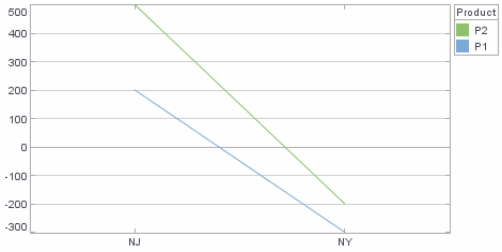
Parameters

boolean	true: independent stack order for each group false: single stack order for all groups

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset =
[["State","Product","Quantity"],["NJ","P1",200],["NJ","P2",
300], ["NY","P1",-300],["NY","P2",100]];
graph = new EGraph();
var elem = new LineElement("State", "Quantity");
var scale = new LinearScale("Quantity");
elem.setColorFrame(new CategoricalColorFrame("Product"));
elem.setStackGroup(true);
elem.setStackNegative(false);
elem.setCollisionModifier(GraphElement.STACK_SYMMETRIC);
graph.addElement(elem);
```



Inherits from

GraphElement: addDim(), addVar(), setCollisionModifier, setColorFrame, setInPlot, setLabelPlacement, setLineFrame, setHint(), setShapeFrame, setSizeFrame, setTextFrame, setTextSpec, setTextureFrame

See Also

DataSet, to use a Data Block or query as the *dataset* source.

LineElement.SETSTARTARROW(BOOLEAN)

Specifies whether an arrow should be drawn at the start of the line (first point).

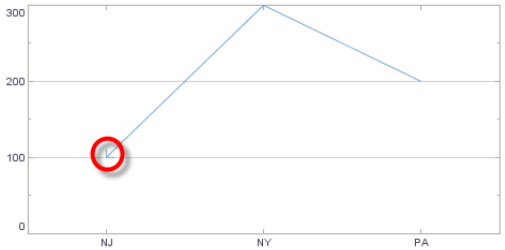
Parameters

boolean	true: draw arrow false: do not draw arrow

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset =
[["State","Quantity"],["NJ",100],["NY",300],["PA",200]];
graph = new EGraph();
elem = new LineElement("State", "Quantity")
elem.setStartArrow(true);
graph.addElement(elem);
```



Inherits from

GraphElement: addDim(), addVar(), setCollisionModifier, setColorFrame, setInPlot, setLabelPlacement, setLineFrame, setHint(), setShapeFrame, setSizeFrame, setTextFrame, setTextSpec, setTextureFrame

See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.5.4 SchemaElement

The SchemaElement object contains user-defined visual elements that can be added to the graph. To create a new SchemaElement object, call the SchemaElement constructor.

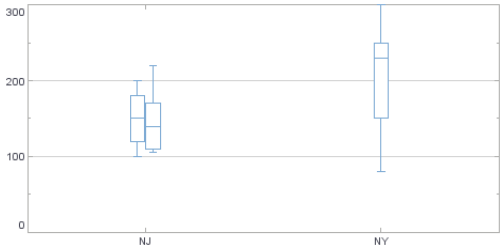
```
var elem = new SchemaElement("State", "Quantity");
```

You can pass a set of field names (e.g., 'State', 'Quantity') to the constructor, or specify these later using the `addDim()` and `addVar()` methods.

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset =
[["State","Hi","Upper","Median","Lower","Lo"],["NJ",200,180
,150,120,100],["NJ",220,170,140,110,105],["NY",300,250,230,
150,80]];
graph = new EGraph();
var elem = new SchemaElement();
elem.addDim("State")
elem.addSchema("Hi","Upper","Median","Lower","Lo")
elem.setPainter(new BoxPainter());
graph.addElement(elem);
```



Full Name

`inetsoft.graph.element.SchemaElement`

SchemaElement: setPainter

Inherits from

GraphElement: addDim(), addVar(), setCollisionModifier, setColorFrame, setInPlot, setLabelPlacement, setLineFrame, setHint(), setShapeFrame, setSizeFrame, setTextFrame, setTextSpec, setTextureFrame

See Also

DataSet, to use a Data Block or query as the *dataset* source.

SchemaElement.SETPAINTER(PAINTER)

Specifies the SchemaPainter object to be used in drawing the visual chart elements.

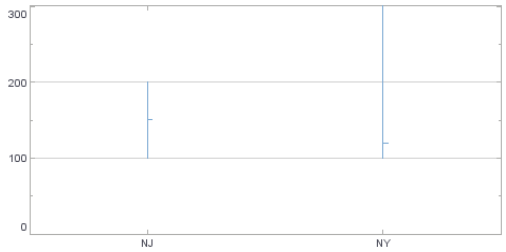
Parameter

painter	BoxPainter: Box and whiskers plot CandlePainter: Candle plot StockPainter: Stock chart

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Hi", "Lo", "Open", "Close"],["NJ",
200, 100, 120, 150],["NY", 300, 100, 200, 120]]
graph = new EGraph();
var elem = new SchemaElement();
elem.addDim("State")
elem.addSchema("Hi", "Close", "Lo")
elem.setPainter(new StockPainter());
graph.addElement(elem);
```



Inherits from

GraphElement: addDim(), addVar(), setCollisionModifier, setColorFrame, setInPlot, setLabelPlacement, setLineFrame, setHint(), setShapeFrame, setSizeFrame, setTextFrame, setTextSpec, setTextureFrame

See Also

DataSet, to use a Data Block or query as the *dataset* source.

SchemaElement.ADDSCHEMA(COL)

Specifies the SchemaPainter object to be used in drawing the visual chart elements.

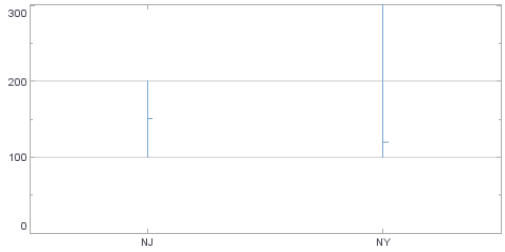
Parameter

col	list of columns for schema binding

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Hi", "Lo", "Open", "Close"],["NJ",
200, 100, 120, 150],["NY", 300, 100, 200, 120]];
graph = new EGraph();
var elem = new SchemaElement();
elem.addDim("State")
elem.addSchema("Hi", "Close", "Lo")
elem.setPainter(new StockPainter());
graph.addElement(elem);
```



Inherits from

GraphElement: addDim(), addVar(), setCollisionModifier, setColorFrame, setInPlot, setLabelPlacement, setLineFrame, setHint(), setShapeFrame, setSizeFrame, setTextFrame, setTextSpec, setTextureFrame

See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.5.5 PointElement

The PointElement object contains the visual elements for a point (scatter) chart. To create a new PointElement object, call the PointElement constructor.

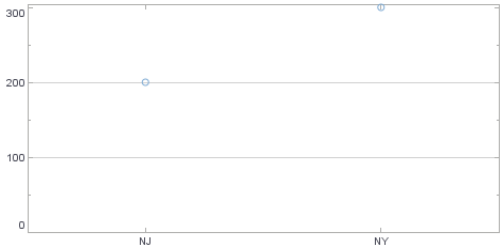
```
var elem = new PointElement("State", "Quantity");
```

You can pass a set of field names (e.g., 'State', 'Quantity') to the constructor, or specify these later using the `addDim()` and `addVar()` methods.

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ", 200], ["NY", 300]];
graph = new EGraph();
var elem = new PointElement("State", "Quantity")
graph.addElement(elem);
```



Full Name

```
inetsoft.graph.element.PointElement
```

Inherits from

GraphElement: addDim(), addVar(), setCollisionModifier, setColorFrame, setInPlot, setLabelPlacement, setLineFrame, setHint(), setShapeFrame, setSizeFrame, setTextFrame, setTextSpec, setTextureFrame

See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.5.6 AreaElement

The AreaElement object contains the visual elements for an area chart. To create a new AreaElement object, call the AreaElement constructor.

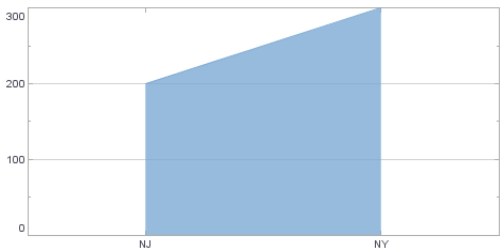
```
var elem = new AreaElement("State", "Quantity");
```

You can pass a set of field names (e.g., 'State', 'Quantity') to the constructor, or specify these later using the `addDim()` and `addVar()` methods.

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ", 200], ["NY", 300]];
graph = new EGraph();
elem = new AreaElement("State", "Quantity");
graph.addElement(elem);
```



Full Name

```
inetsoft.graph.element.AreaElement
```

Inherits from

LineElement: setClosed, setEndArrow, setStackGroup, setStackNegative, setStartArrow
GraphElement: addDim(), addVar(), setCollisionModifier, setColorFrame, setInPlot, setLabelPlacement, setLineFrame, setHint(), setShapeFrame, setSizeFrame, setTextFrame, setTextSpec, setTextureFrame

See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.6 Chart Coordinates & Scaling

This section discusses objects related to setting chart coordinates and their scaling properties. See *Changing Chart Scaling* for a tutorial introduction.

CR.6.1 Coordinate

The Coordinate object contains the coordinates against which data can be represented.

Full Name

```
inetsoft.graph.coord.Coordinate
```

Coordinate: reflect, rotate, setExtent, transpose

Created by

PolarCoord
RectCoord
ParallelCoord
FacetCoord

Coordinate.REFLECT(VERT)

Reflect the coordinates about the vertical or horizontal axis.

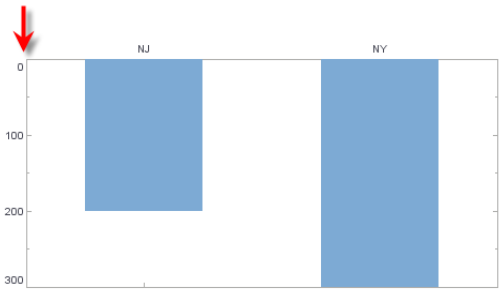
Parameter

vert	true: reflect about horizontal axis false: reflect about vertical axis

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ", 200], ["NY", 300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var sscale = new CategoricalScale("State");
var qscale = new LinearScale("Quantity");
var coord = new RectCoord(sscale,qscale);
coord.reflect(true);
graph.setCoordinate(coord);
graph.addElement(elem);
```



See Also

DataSet, to use a Data Block or query as the *dataset* source.

Coordinate.ROTATE(VALUE)

Rotate the axes by the specified angle.

Parameter

value an Angle in degrees

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["Direction", "Score"],[(Math.PI/
2),20],[(Math.PI/4),30],[(Math.PI),35]];
graph = new EGraph();
var elem = new PointElement("Direction", "Score");
var xscale = new LinearScale("Direction");
var yscale = new LinearScale("Score");
yscale.setMin(0);
yscale.setMax(40);
xscale.setMin(0);
xscale.setMax(1.95*Math.PI);
xscale.setIncrement(Math.PI/8);
var rect = new RectCoord(xscale,yscale);
var polar = new PolarCoord(rect);
polar.rotate(45)
polar.setType(PolarCoord.THETA);
graph.setCoordinate(polar);
graph.addElement(elem);
```



See Also

DataSet, to use a Data Block or query as the *dataset* source.

Coordinate.SETEXTENT(MINX, MINY, MAXX, MAXY)

Sets the extent of a geographical coordinate system (for a map-type chart) to the specified latitudes and longitudes.

Parameter

minX	the minimum longitude
minY	the minimum latitude
maxX	the maximum longitude
maxY	the maximum latitude

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

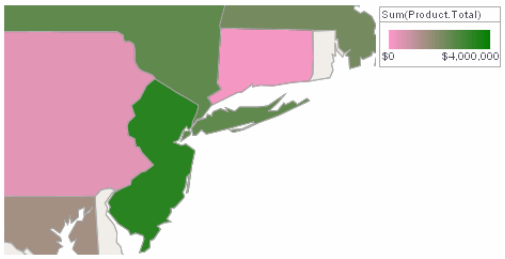
Example (Report or Viewsheet)

Create a map of the United States. (See *Creating a Map Chart* in *Advanced Dashboard Design* or *Binding a Chart to Geographical Data (Map)* in *Advanced Report Design* for more information.) To restrict the range of the map to the region between Washington and Boston, first obtain the latitude and longitude of these cities:

- Washington: 38.895111 latitude, -77.036667 longitude
- Boston: 42.358056 latitude, -71.063611 longitude

Add the following chart script to limit the extent of the map to this range. Note that the *x*-axis corresponds to longitude, and the *y*-axis corresponds to latitude.

```
graph.getCoordinate().setExtent(-77.036667, 38.895111,-
71.063611, 42.358056)
graph.getElement(0).setInPlot(false)
```



The `setInPlot(false)` function allows the chart to display only the specified portion of the map. Otherwise, the chart will force the entire map to display, regardless of the specified extent. See *GraphElement.setInPlot(boolean)* for more information.

Coordinate.SETFULLMAP(BOOLEAN)

Specifies, when 'true', that the entire map should be displayed within the element's borders. Otherwise, only regions of the map representing data are displayed, and other regions are cropped.

Parameter

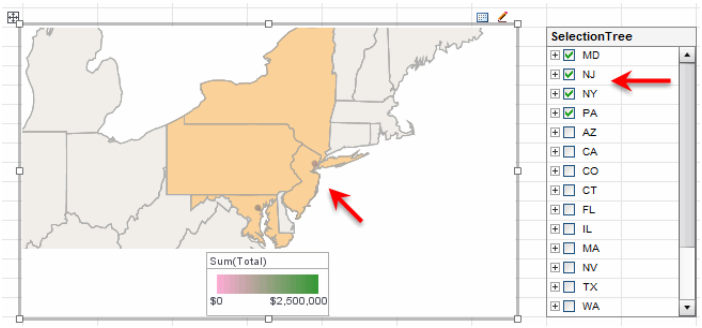
boolean (default)	`true: fit entire map in chart borders` `false: fit only data regions in chart borders`

`Note: Script that modifies 'graph' should be placed at the element level. See` *`Adding Element-Level Script`* `in` *`Report Scripting`* `and` *`Adding Component Script`* `in` *`Dashboard Scripting`* `for more information.`

Example (Viewsheet)

Create the map described in the example for *drillMembers* in *Advanced Dashboard Scripting*. To zoom the map to display only the selected regions, add the following script to the Chart component:

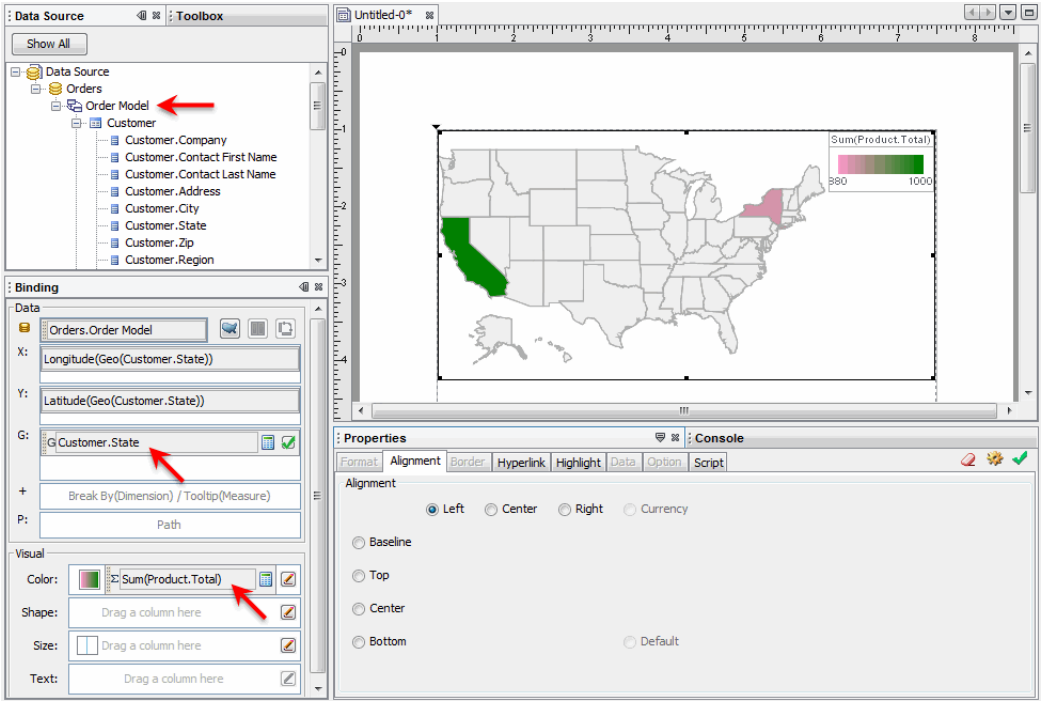
```
var zoomed = SelectionTree1.selectedObjects.length > 0;
graph.getCoordinate().setFullMap(!zoomed);
graph.getElement(0).setInPlot(!zoomed);
```



The `setInPlot(false)` function allows the chart to display only the specified portion of the map. Otherwise, the chart will force the entire map to display regardless of the `setFullMap` setting. See *GraphElement.setInPlot(boolean)* for more information.

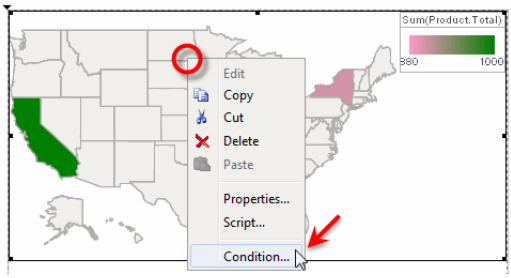
Example (Report)

Create a report, as shown below, containing a US map based on the sample 'Order Model' data model. (See *Binding a Chart to Geographical Data (Map)* in *Advanced Report Design* for more information on how to create a map.)

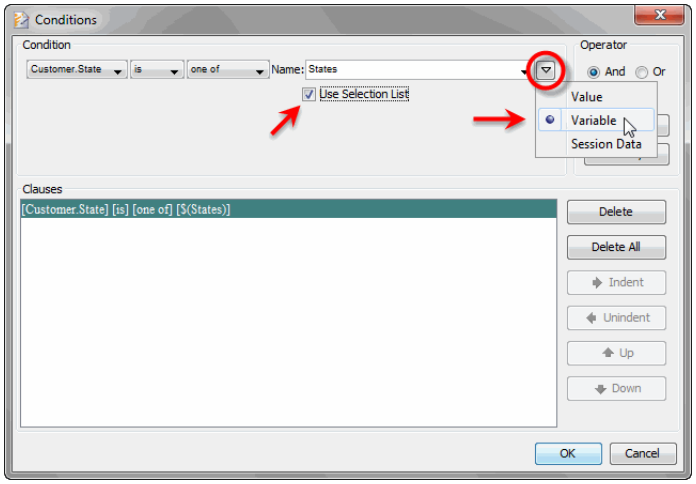


Add a parameterized condition to the map. To do this, follow the steps below:

1. Right-click on the map, and select ‘Condition’ from the context menu. This opens the ‘Condition’ dialog box.



2. Use the menus to enter the following parameterized condition. Make sure to select the ‘Variable’ option from the right-side pop-up menu, and enable the ‘Use Selection List’ option.



3. Press ‘Append’ to add the condition, and then press ‘OK’.

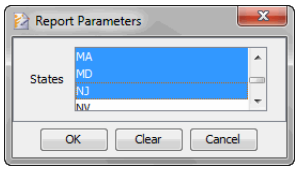
4. Right-click on the map and select ‘Script’ from the context menu to open the Script Editor. Add the following script:

   ```
   graph.getCoordinate().setFullMap(false);
   graph.getElement(0).setInPlot(false);
   ```

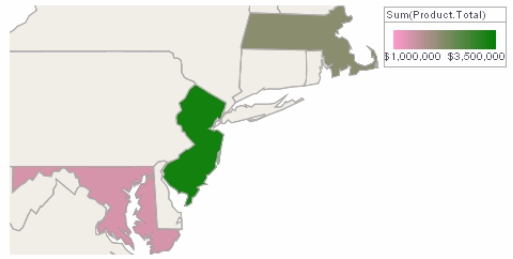
   The `setInPlot(false)` function allows the chart to display only the specified portion of the map. Otherwise, the chart will force the entire map to display regardless of the `setFullMap` setting. See *GraphElement.setInPlot(boolean)* for more information.

5. Press ‘Save and Close’ to close the Script Editor.

6. Preview the report and select a set of nearby states when prompted for the ‘States’ parameter.



Observe that the resulting map is cropped to display only data from the selected geographical regions.



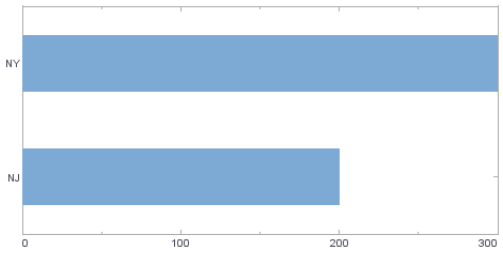
Coordinate.TRANSPOSE()

Interchanges the axes. For example, in a rectangular coordinate system, the X-axis becomes the Y-axis, and the Y-axis becomes the X-axis.

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ", 200], ["NY", 300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var sscale = new CategoricalScale("State");
var qscale = new LinearScale("Quantity");
var coord = new RectCoord(sscale,qscale);
coord.transpose()
graph.setCoordinate(coord);
graph.addElement(elem);
```



See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.6.2 PolarCoord

The PolarCoord object contains polar coordinates against which data can be represented. To create a PolarCoord object, call the PolarCoord constructor:

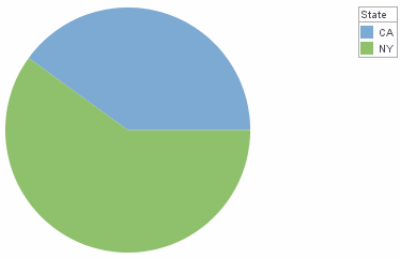
```
var polar = new PolarCoord(rect);
```

You can pass a *RectCoord* object to the constructor (e.g., 'rect'), or specify this later using the *PolarCoord.setCoordinate(coord)* property.

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State","Quantity"], ["CA",200], ["NY",300]];
graph = new EGraph();
var elem = new IntervalElement("Quantity");
var yscale = new LinearScale("Quantity");
var rect = new RectCoord(null, yscale);
var polar = new PolarCoord(rect);
var frame = new CategoricalColorFrame();
var range = new StackRange();
polar.setType(PolarCoord.THETA);
frame.setField("State");
elem.setColorFrame(frame);
elem.setCollisionModifier(GraphElement.MOVE_STACK);
yscale.setScaleRange(range);
var spec = new AxisSpec();
spec.setLabelVisible(false);
spec.setTickVisible(false);
spec.setLineVisible(false);
yscale.setAxisSpec(spec);
graph.setCoordinate(polar);
graph.addElement(elem);
```



Full Name

```
inetsoft.graph.coord.PolarCoord
```

PolarCoord: setCoordinate, setType

Inherits from

Coordinate: reflect, rotate, setExtent, transpose

See Also

DataSet, to use a Data Block or query as the *dataset* source.
Polar Coordinates, for a walkthrough of polar coordinates.

PolarCoord.SETTYPE(VALUE)

Specifies the type of polar transformation.

Parameter

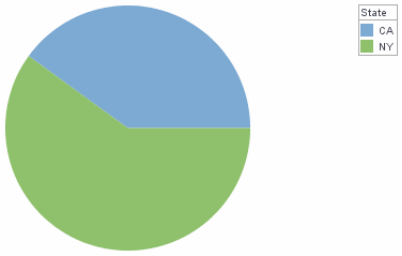
value	PolarCoord.THETA	(use only angle)
	PolarCoord.THETA_RHO	(use angle and radius)
	PolarCoord.RHO	(use only radius)

PolarCoord.PLUS (angle & radius, with hole)

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State","Quantity"], ["CA",200], ["NY",300]];
graph = new EGraph();
var elem = new IntervalElement("Quantity");
var yscale = new LinearScale("Quantity");
var rect = new RectCoord(null, yscale);
var polar = new PolarCoord(rect);
var frame = new CategoricalColorFrame();
var range = new StackRange();
polar.setType(PolarCoord.THETA);
frame.setField("State");
elem.setColorFrame(frame);
elem.setCollisionModifier(GraphElement.MOVE_STACK);
yscale.setScaleRange(range);
var spec = new AxisSpec();
spec.setLabelVisible(false);
spec.setTickVisible(false);
spec.setLineVisible(false);
yscale.setAxisSpec(spec);
graph.setCoordinate(polar);
graph.addElement(elem);
```



Inherits from

Coordinate: reflect, rotate, setExtent, transpose

See Also

DataSet, to use a Data Block or query as the *dataset* source.

PolarCoord.SETCOORDINATE(COORD)

Specifies the rectangular coordinates on which the polar coordinates should be based.

Parameter

coord a *RectCoord* object

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["Direction", "Score"], [(Math.PI/2),20],
[(Math.PI/4),30], [(Math.PI),35]];
graph = new EGraph();
var elem = new PointElement("Direction", "Score");
var xscale = new LinearScale("Direction");
var yscale = new LinearScale("Score");
yscale.setMin(0);
yscale.setMax(40);
xscale.setMin(0);
xscale.setMax(1.95*Math.PI);
xscale.setIncrement(Math.PI/8);
var rect = new RectCoord(xscale,yscale);
var polar = new PolarCoord();
polar.setCoordinate(rect);
graph.setCoordinate(polar);
graph.addElement(elem);
```

Inherits from

Coordinate: reflect, rotate, setExtent, transpose

See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.6.3 RectCoord

The RectCoord object contains rectangular coordinates against which data can be represented. To create a RectCoord object, pass a pair of *Scale* objects to the RectCoord constructor:

```
var rect = new RectCoord(xscale,yscale);
```

You can pass a set of *Scale* objects (e.g., 'xscale', 'yscale', etc.) to the constructor, or specify this later using the *RectCoord.setXScale(scale)* and *RectCoord.setYScale(scale)* properties.

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State","Quantity"], ["NJ",200], ["NY",300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var sscale = new CategoricalScale("State");
var qscale = new LinearScale("Quantity");
var coord = new RectCoord(sscale,qscale);
graph.setCoordinate(coord);
graph.addElement(elem);
```

Full Name

```
inetsoft.graph.coord.RectCoord
```

RectCoord: setXScale, setYScale, setYScale2

Inherits from
Coordinate: reflect, rotate, setExtent, transpose

See Also
DataSet, to use a Data Block or query as the *dataset* source.
Rectangular Coordinates, for a walkthrough of rectangular coordinates.

RectCoord.SETXSCALE(SCALE)

Specifies the scale for the X-axis. This can also be specified as an argument to the constructor.

Type

scale	a *Scale* object

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State","Quantity"], ["NJ",200], ["NY",300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var sscale = new CategoricalScale("State");
var qscale = new LinearScale("Quantity");
var coord = new RectCoord();
coord.setXScale(sscale);
coord.setYScale(qscale);
graph.setCoordinate(coord);
graph.addElement(elem);
```

Inherits from
Coordinate: reflect, rotate, setExtent, transpose

See Also
DataSet, to use a Data Block or query as the *dataset* source.

RectCoord.SETYSCALE(SCALE)

Specifies the scale for the Y-axis. This can also be specified as an argument to the constructor.

Type

scale	a *Scale* object

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State","Quantity"], ["NJ",200], ["NY",300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
```

```
var sscale = new CategoricalScale("State");
var qscale = new LinearScale("Quantity");
var coord = new RectCoord();
coord.setXScale(sscale);
coord.setYScale(qscale);
graph.setCoordinate(coord);
graph.addElement(elem);
```

Inherits from

Coordinate: reflect, rotate, setExtent, transpose

See Also

DataSet, to use a Data Block or query as the *dataset* source.

RectCoord.SETYSCALE2(SCALE)

Specifies the scale for the secondary (right-side) Y-axis.

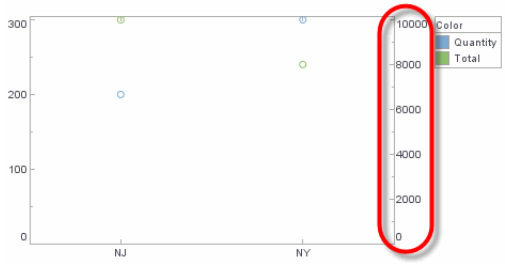
Type

scale a *Scale* object

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State","Quantity","Total"], ["NJ",200,10000],
["NY",300,8000]];
graph = new EGraph();
var elem = new PointElement("State", "Quantity");
var elem2 = new PointElement("State", "Total");
//elem2.setColorFrame(new
StaticColorFrame(java.awt.Color.red));
var cframe = new CategoricalColorFrame();
cframe.init("Quantity", "Total");
elem.setColorFrame(cframe);
elem2.setColorFrame(cframe);
var sscale = new CategoricalScale("State");
var qscale = new LinearScale("Quantity");
var qscale2 = new LinearScale("Total");
var coord = new RectCoord();
coord.setXScale(sscale);
coord.setYScale(qscale);
coord.setYScale2(qscale2);
graph.setCoordinate(coord);
graph.addElement(elem);
graph.addElement(elem2);
```



Inherits from
Coordinate: reflect, rotate, setExtent, transpose

See Also
DataSet, to use a Data Block or query as the *dataset* source.

CR.6.4 Rect25Coord

The Rect25Coord object is the same as the *RectCoord* object, but creates a 3D effect for elements. To create a Rect25Coord object, call the object constructor:

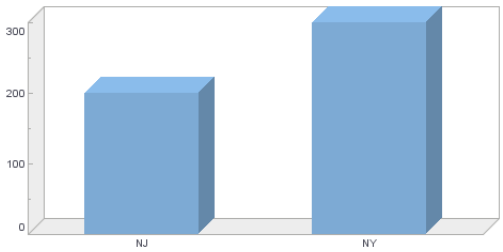
```
var rect = new Rect25Coord(xscale,yscale);
```

You can pass a set of *Scale* objects (e.g., 'xscale', 'yscale', etc.) to the constructor, or specify these later using the inherited *RectCoord.setXScale(scale)* and *RectCoord.setYScale(scale)* properties.

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ", 200], ["NY", 300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var sscale = new CategoricalScale("State");
var qscale = new LinearScale("Quantity");
var coord = new Rect25Coord(sscale,qscale);
graph.setCoordinate(coord);
graph.addElement(elem);
```



Full Name

```
inetsoft.graph.coord.Rect25Coord
```

Inherits from
RectCoord: setXScale, setYScale, setYScale2
Coordinate: reflect, rotate, setExtent, transpose

See Also
DataSet, to use a Data Block or query as the *dataset* source.

CR.6.5 ParallelCoord

The ParallelCoord object contains parallel coordinates against which data can be represented. To create a ParallelCoord object, pass a set of *Scale* objects to the ParallelCoord constructor:

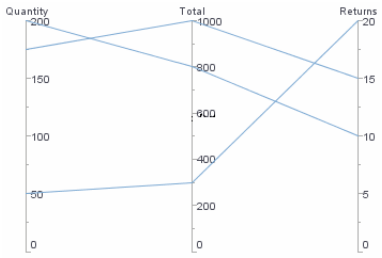
```
var rect = new ParallelCoord(scale1,scale2,...);
```

You can pass a set of *Scale* objects (e.g., 'scale1', 'scale2', etc.) to the constructor, or specify this later using the *ParallelCoord.setScales(scales)* property.

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset =
[["Quantity","Total","Returns"],[200,800,10],[175,1000,15],
[50,300,20]];
graph = new EGraph();
var elem = new LineElement();
var qscale = new LinearScale("Quantity");
var tscale = new LinearScale("Total");
var rscale = new LinearScale("Returns");
var coord = new ParallelCoord(qscale,tscale,rscale);
elem.addDim("Quantity");
elem.addDim("Total");
elem.addDim("Returns");
graph.addElement(elem);
graph.setCoordinate(coord);
```



Full Name

```
inetsoft.graph.coord.ParallelCoord
```

ParallelCoord: setScales

Inherits from

Coordinate: reflect, rotate, setExtent, transpose

See Also

DataSet, to use a Data Block or query as the *dataset* source.

Appendix SC.12.2, *Polar Coordinates*, for a walkthrough example of parallel coordinates.

ParallelCoord.SETSCALES(SCALES)

Specifies the set of parallel scales to use. These can also be specified as arguments to the constructor.

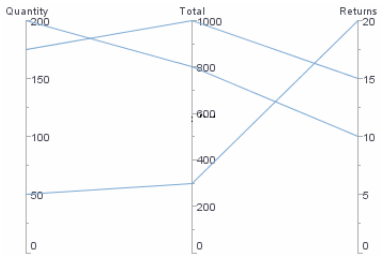
Parameter

scales Array of *Scale* objects

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset =
[["Quantity","Total","Returns"],[200,800,10],[175,1000,15],
[50,300,20]];
graph = new EGraph();
var elem = new LineElement();
var qscale = new LinearScale("Quantity");
var tscale = new LinearScale("Total");
var rscale = new LinearScale("Returns");
var coord = new ParallelCoord();
coord.setScales([qscale,tscale,rscale]);
elem.addDim("Quantity");
elem.addDim("Total");
elem.addDim("Returns");
graph.addElement(elem);
graph.setCoordinate(coord);
```



Inherits from

Coordinate: reflect, rotate, setExtent, transpose

See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.6.6 TriCoord

The TriCoord object contains triangular coordinates against which data can be represented. Although the triangular coordinate system contains three axes, it represents only two independent dimensions, and the three measures must sum to the maximum value of the scale. This makes the triangular coordinates effective for representing proportions of a fixed total.

Note: The three measures must sum to the maximum value of the scale.

To create a TriCoord object, call the TriCoord constructor:

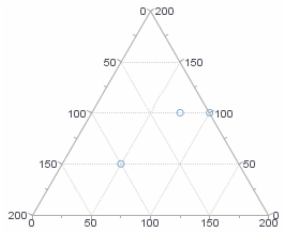
```
var coord = new TriCoord(qscale);
```

You can pass a *Scale* object (e.g., 'qscale') to the constructor, or specify this later using the *TriCoord.setScale(scale)* property.

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset =
[["Quantity","Total","Returns"],[50,50,100],[75,100,25],[10
0,100,0]];
graph = new EGraph();
var elem = new PointElement();
var qscale = new LinearScale("Quantity");
qscale.setMin(0);
qscale.setMax(200);
var spec = new AxisSpec();
spec.setGridStyle(Chart.DOT_LINE);
qscale.setAxisSpec(spec);
var coord = new TriCoord(qscale);
elem.addDim("Quantity");
elem.addDim("Total");
elem.addVar("Returns");
graph.addElement(elem);
graph.setScale("Quantity", qscale);
graph.setScale("Total", qscale);
graph.setScale("Returns", qscale);
graph.setCoordinate(coord);
```



The first dimension ('Quantity') is represented on the bottom axis, the second dimension ('Total') on the right axis, and the third dimension ('Returns') on the left axis. Note that the third dimension must be added using addVar() rather than addDim().

Full Name

```
inetsoft.graph.coord.TriCoord
```

TriCoord: setScale

Inherits from

Coordinate: reflect, rotate, setExtent, transpose

See Also

DataSet, to use a Data Block or query as the *dataset* source.

TriCoord.SETSCALE(SCALE)

Specifies the scale to use for all three axes. This can also be specified as an argument to the constructor.

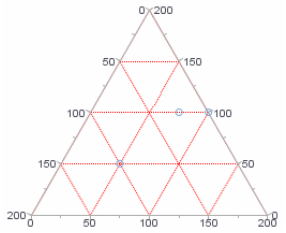
Parameter

scale a *Scale* object

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset =
[["Quantity","Total","Returns"],[50,50,100],[75,100,25],[10
0,100,0]];
graph = new EGraph();
var elem = new PointElement();
var qscale = new LinearScale("Quantity");
qscale.setMin(0);
qscale.setMax(200);
var spec = new AxisSpec();
spec.setGridStyle(Chart.DOT_LINE);
spec.setGridColor(java.awt.Color.red);
qscale.setAxisSpec(spec);
var coord = new TriCoord();
coord.setScale(qscale);
elem.addDim("Quantity");
elem.addDim("Total");
elem.addVar("Returns");
graph.addElement(elem);
graph.setScale("Quantity", qscale);
graph.setScale("Total", qscale);
graph.setScale("Returns", qscale);
graph.setCoordinate(coord);
```



Inherits from

Coordinate: reflect, rotate, setExtent, transpose

See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.6.7 FacetCoord

The FacetCoord object contains a set of inner and outer coordinates on which multidimensional data can be represented as nested charts. To create a FacetCoord object, call the FacetCoord constructor:

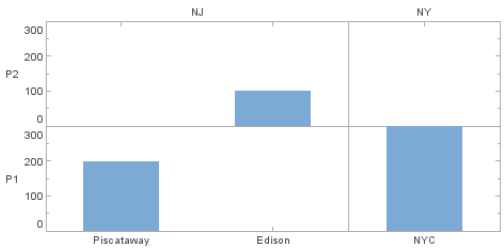
```
var rect = new FacetCoord(outerCoord,innerCoord);
```

You can pass a pair of *RectCoord* objects (e.g., 'outerCoord', 'innerCoord', etc.) to the constructor, or specify these later using the *FacetCoord.setInnerCoordinates(coord)* and *FacetCoord.setOuterCoordinate(coord)* properties.

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "City", "Product", "Quantity"],["NJ",
"Piscataway", "P1", 200],["NJ", "Edison", "P2", 100],["NY",
"NYC", "P1", 300]];
graph = new EGraph();
var elem = new IntervalElement("City", "Quantity");
var state = new CategoricalScale("State");
var city = new CategoricalScale("City");
var product = new CategoricalScale("Product");
var quantity = new LinearScale("Quantity");
var inner = new RectCoord(city, quantity);
var outer = new RectCoord(state, product);
var coord = new FacetCoord(outer,inner);
graph.setCoordinate(coord);
graph.addElement(elem);
```



Full Name

```
inetsoft.graph.coord.FacetCoord
```

FacetCoord: setInnerCoordinates, setOuterCoordinate

Inherits from

Coordinate: reflect, rotate, setExtent, transpose

See Also

DataSet, to use a Data Block or query as the *dataset* source.

Appendix SC.12.4, *Facet Coordinates*, for a walkthrough of facet coordinates.

FacetCoord.SETINNERCOORDINATES(COORD)

Specifies a *RectCoord* object, or array of *RectCoord* objects, to use for the inner coordinates of the facet graph. If an array is specified, each inner coordinate set is plotted independently. The 'innerCoordinates' property can also be specified as an argument to the constructor.

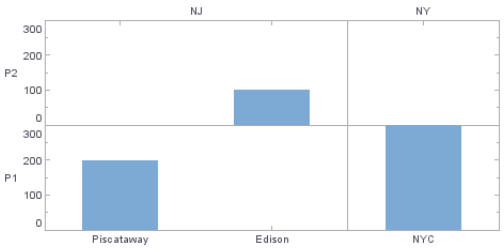
Parameter

coord Array of *RectCoord* objects

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "City", "Product", "Quantity"],["NJ",
"Piscataway", "P1", 200],["NJ", "Edison", "P2", 100],["NY",
"NYC", "P1", 300]];
graph = new EGraph();
var elem = new IntervalElement("City", "Quantity");
var state = new CategoricalScale("State");
var city = new CategoricalScale("City");
var product = new CategoricalScale("Product");
var quantity = new LinearScale("Quantity");
var inner = new RectCoord(city, quantity);
var outer = new RectCoord(state, product);
var coord = new FacetCoord();
coord.setInnerCoordinates([inner]);
coord.setOuterCoordinate(outer);
graph.setCoordinate(coord);
graph.addElement(elem);
```



Inherits from

Coordinate: reflect, rotate, setExtent, transpose

See Also

DataSet, to use a Data Block or query as the *dataset* source.

FacetCoord.SETOUTERCOORDINATE(COORD)

Specifies the rectangular coordinate set to use for the outer coordinates of the facet graph. The *RectCoord* object used for outer coordinates generally has categorical scales on both axes. The 'outerCoordinate' can also be specified as an argument to the constructor.

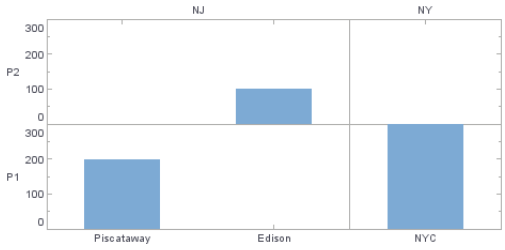
Parameter

coord	a *RectCoord* object

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "City", "Product", "Quantity"],["NJ",
"Piscataway", "P1", 200],["NJ", "Edison", "P2", 100],["NY",
"NYC", "P1", 300]];
graph = new EGraph();
var elem = new IntervalElement("City", "Quantity");
var state = new CategoricalScale("State");
var city = new CategoricalScale("City");
var product = new CategoricalScale("Product");
var quantity = new LinearScale("Quantity");
var inner = new RectCoord(city, quantity);
var outer = new RectCoord(state, product);
var coord = new FacetCoord();
coord.setInnerCoordinates([inner]);
coord.setOuterCoordinate(outer);
graph.setCoordinate(coord);
graph.addElement(elem);
```



Inherits from

Coordinate: reflect, rotate, setExtent, transpose

See Also

DataSet, to use a Data Block or query as the *dataset* source.

FacetCoord.SETVERTICAL(BOOLEAN)

Specifies whether the inner coordinates are arranged vertically or horizontally.

Parameter

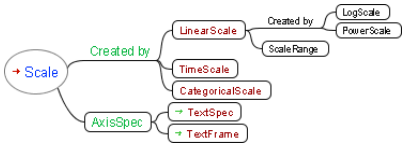
boolean	true: stack vertically (default) false: stack horizontally

Inherits from

Coordinate: reflect, rotate, setExtent, transpose

CR.6.8 Scale

The Scale object contains a scale defining the measurement of a dimension.



Full Name

`inetsoft.graph.scale.Scale`

Scale: init, setAxisSpec, setFields, setScaleOption

Created by

LinearScale
LogScale
PowerScale
TimeScale
CategoricalScale

Scale.INIT(DATASET)

Forces the immediate computation of the automatic scale attributes. The `init()` method is only required if you need to access automatically-computed scale attributes within your chart script. These attributes are not available within a script until the scale is initialized via the `init()` method.

Parameter

dataset The Chart dataset attribute

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ",200], ["NY",300]];
graph = new EGraph();
var qscale = new LinearScale("Quantity");
var elem = new IntervalElement("State", "Quantity")
qscale.init(dataset)
var defaultMax = qscale.getMax();
qscale.setMax(defaultMax + defaultMax/2);
graph.setScale("Quantity", qscale);
graph.addElement(elem);
```

See Also

DataSet, to use a Data Block or query as the *dataset* source.

Scale.SETAXISSPEC(SPEC)

Specifies the axis properties for the scale.

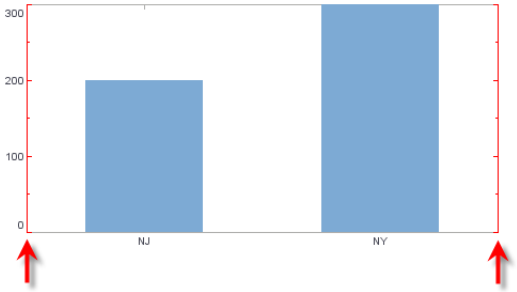
Parameter

spec *AxisSpec*

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ",200], ["NY",300]];
graph = new EGraph();
var qscale = new LinearScale("Quantity");
var elem = new IntervalElement("State", "Quantity")
var spec = new AxisSpec();
spec.setLineColor(java.awt.Color(0xff0000));
qscale.setAxisSpec(spec);
graph.setScale("Quantity", qscale);
graph.addElement(elem);
```



See Also

DataSet, to use a Data Block or query as the *dataset* source.

Scale.SETDATAFIELDS(ARR)

Specifies the fields to use for initializing the scale; i.e., determining the maximum and minimum values. If left unspecified, the values in the *Scale.setFields(field)* property are used for this purpose.

Parameter

arr Array of Strings

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State","Quantity","Total"], ["NJ",100,0],
["NY",1500,30000]];
graph = new EGraph();
var qscale = new LogScale();
var elem = new IntervalElement("State", "Quantity");
qscale.setDataFields(["Total"]);
graph.addElement(elem)
graph.setScale("Quantity", qscale);
```

See Also

DataSet, to use a Data Block or query as the *dataset* source.

Scale.SETFIELDS(FIELD)

Specifies the fields to which the scale should be applied.

Parameter

field	a String containing a column name

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State","Quantity"],["NJ",100],["NY",4000]];
graph = new EGraph();
var qscale = new LogScale();
var elem = new IntervalElement("State", "Quantity");
qscale.setFields(["Quantity"]);
var coord = new RectCoord(new CategoricalScale("State"),
qscale)
graph.setCoordinate(coord);
graph.addElement(elem);
```

See Also

DataSet, to use a Data Block or query as the *dataset* source.

Scale.SETSCALEOPTION(VALUE)

Specifies a scaling option for the default scaling. The `Scale.TICKS` and `Scale.ZERO` options determine the maximum and minimum values that are used to calculate the scale range.

The `Scale.TICKS` option uses the maximum and minimum tick values (i.e., rounded numbers) rather than the maximum and minimum data values. The `Scale.ZERO` option uses zero as the minimum rather than the minimum data value (if positive). To combine multiple options, use the pipe ("or") operator:

```
qscale.setScaleOption(Scale.ZERO | Scale.TICKS);
```

Parameter

value	Scale.RAW	(no modification)
	Scale.NO_NULL	(remove NULL-data gaps in scale)
	Scale.TICKS	(use ticks in scale range calculation)
	Scale.ZERO	(use zero in scale range)

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ", 200], ["NY", 300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
```

```
var sscale = new CategoricalScale("State");
var qscale = new LinearScale("Quantity");
var coord = new RectCoord(sscale,qscale);
qscale.setScaleOption(Scale.ZERO);
graph.setCoordinate(coord);
graph.addElement(elem);
```

See Also

DataSet, to use a Data Block or query as the *dataset* source.

Scale.SETSHAREDRANGE(BOOLEAN)

For a *FacetCoord*, specifies whether the same scale range should be used across all sub-graphs in a facet, or whether a sub-graph's scaling should be shared only with sub-graphs in the same row (for Y-axis scaling) or same column (for X-axis scaling).

Parameter

value	true	share scaling across all sub-graphs
	false	share scaling by row and column

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "City", "Product", "Quantity"],["NJ",
"Piscataway", "P1", 200],["NJ", "Edison", "P2", 100],["NY",
"NYC", "P1", 300]];
graph = new EGraph();
var elem = new IntervalElement("City", "Quantity");
var state = new CategoricalScale("State");
var city = new CategoricalScale("City");
var product = new CategoricalScale("Product");
var quantity = new LinearScale("Quantity");
quantity.setSharedRange(false)
var inner = new RectCoord(city, quantity);
var outer = new RectCoord(state, product);
var coord = new FacetCoord(outer,inner);
graph.setCoordinate(coord);
graph.addElement(elem);
```

See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.6.9 LinearScale

The LinearScale object contains a linear scale, i.e., a scale that linearly maps numerical values to physical attributes. To create a LinearScale object, call the LinearScale constructor.

```
var qscale = new LinearScale('Last Year','This Year');
```

You can pass the names of the fields (e.g., 'Last Year', 'This Year') for which the scale should be generated to the constructor, or specify these later using the inherited *Scale.setFields(field)* property.

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State","Quantity"], ["NJ",200], ["NY",300]];
graph = new EGraph();
var qscale = new LinearScale();
qscale.setFields(["Quantity"]);
var elem = new IntervalElement("State", "Quantity")
graph.setScale("Quantity", qscale);
graph.addElement(elem);
```

Full Name

```
inetsoft.graph.scale.LinearScale
```

Inherits from

Scale: init, setAxisSpec, setFields, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

LinearScale: setIncrement, setMax, setMin, setMinorIncrement, setReversed, setScaleRange

LinearScale.SETINCREMENT(VALUE)

Specifies the interval between values on the scale. When the scale is applied to an axis, this is the interval between values displayed on the axis.

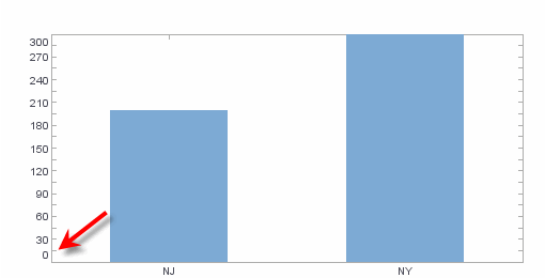
Parameter

value	Number specifying interval between values

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ",200], ["NY",300]];
graph = new EGraph();
var qscale = new LinearScale("Quantity");
var elem = new IntervalElement("State", "Quantity")
qscale.setIncrement(15);
graph.setScale("Quantity", qscale);
graph.addElement(elem);
```



Inherits from

Scale: init, setAxisSpec, setFields, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

LinearScale.SETMAX(VALUE)

Specifies the maximum value of the scale.

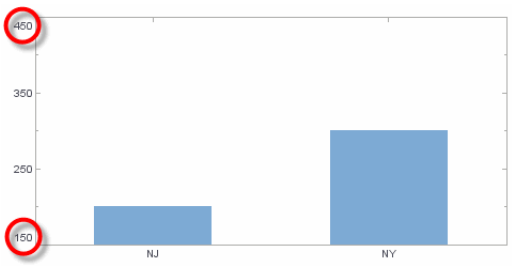
Parameter

value	Number specifying the maximum scale value

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ",200], ["NY",300]];
graph = new EGraph();
var qscale = new LinearScale("Quantity");
var elem = new IntervalElement("State", "Quantity")
qscale.setMin(150);
qscale.setMax(450);
graph.setScale("Quantity", qscale);
graph.addElement(elem);
```



If you set the maximum scale value smaller than the largest data value (so as to crop the data), you should also set `elem.setInPlot(false)`. Otherwise, the chart will force the data to appear in its entirety, resulting in a partially unlabeled axis.

Inherits from
Scale: init, setAxisSpec, setFields, setScaleOption

See Also
DataSet, to use a Data Block or query as the *dataset* source.

LinearScale.SETMIN(VALUE)

Specifies the minimum value of the scale.

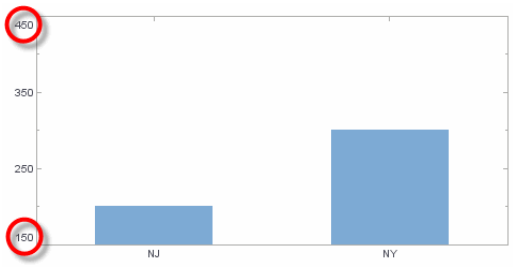
Parameter

value	Number specifying the minimum scale value

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ",200], ["NY",300]];
graph = new EGraph();
var qscale = new LinearScale("Quantity");
var elem = new IntervalElement("State", "Quantity")
qscale.setMin(150);
qscale.setMax(450);
graph.setScale("Quantity", qscale);
graph.addElement(elem);
```



Inherits from
Scale: init, setAxisSpec, setFields, setScaleOption

See Also
DataSet, to use a Data Block or query as the *dataset* source.

LinearScale.SETMINORINCREMENT(VALUE)

Specifies the minor interval between values on the scale. When the scale is applied to an axis, this is the interval between tick marks displayed on the axis.

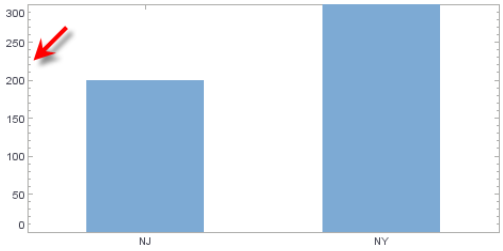
Parameter

value	Number specifying interval between ticks

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ",200], ["NY",300]];
graph = new EGraph();
var qscale = new LinearScale("Quantity");
qscale.setIncrement(50);
var elem = new IntervalElement("State", "Quantity")
qscale.setMinorIncrement(10);
graph.setScale("Quantity", qscale);
graph.addElement(elem);
```



Inherits from

Scale: init, setAxisSpec, setFields, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

LinearScale.SETREVERSED(BOOLEAN)

Specifies the orientation of the scale, either with value increasing from bottom to top (default), or from top to bottom.

Parameter

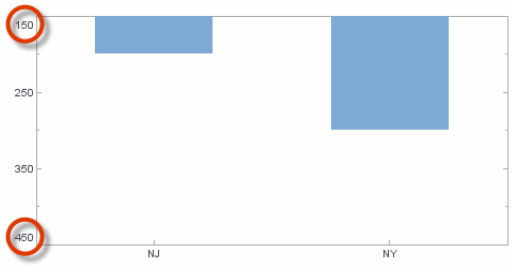
boolean	true: value increases from top to bottom false: value increases from bottom to top

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ",200], ["NY",300]];
graph = new EGraph();
var qscale = new LinearScale("Quantity");
var elem = new IntervalElement("State", "Quantity")
qscale.setMin(150);
qscale.setMax(450);

// Place setReversed() after setMin() and setMax()
qscale.setReversed(true);
graph.setScale("Quantity", qscale);
graph.addElement(elem);
```



Inherits from
Scale: init, setAxisSpec, setFields, setScaleOption

See Also
DataSet, to use a Data Block or query as the *dataset* source.

LinearScale.SETSCALERANGE(RANGE)

Specifies the calculation strategy for finding scale range.

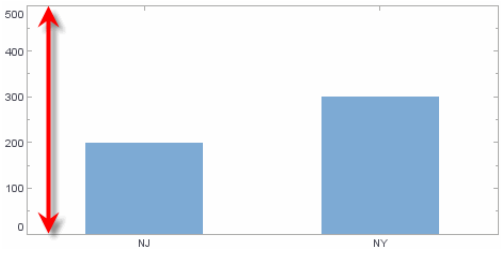
Parameter

range a *ScaleRange* object

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ",200], ["NY",300]];
graph = new EGraph();
var qscale = new LinearScale("Quantity");
var elem = new IntervalElement("State", "Quantity")
qscale.setScaleRange(new StackRange());  // adds 200+300
graph.setScale("Quantity", qscale);
graph.addElement(elem);
```



Inherits from
Scale: init, setAxisSpec, setFields, setScaleOption

See Also
DataSet, to use a Data Block or query as the *dataset* source.

CR.6.10 LogScale

The LogScale object contains a logarithmic scale, i.e., a scale that logarithmically maps numerical data values to physical attributes (axis position, element size, etc.). To create a LogScale object, call the LogScale constructor.

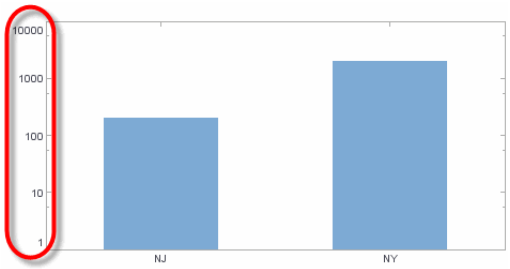
```
var qscale = new LogScale('Last Year', 'This Year');
```

You can pass the names of the fields (e.g., 'Last Year', 'This Year') for which the scale should be generated to the constructor, or specify these later using the inherited *Scale.setFields(field)* property.

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State","Quantity"], ["NJ",200], ["NY",2000]];
graph = new EGraph();
var qscale = new LogScale();
qscale.setFields(["Quantity"]);
var elem = new IntervalElement("State", "Quantity")
graph.setScale("Quantity", qscale);
graph.addElement(elem);
```



Full Name

```
inetsoft.graph.scale.LogScale
```

LogScale: setBase

Inherits from

LinearScale: setIncrement, setMax, setMin, setMinorIncrement, setReversed, setScaleRange
Scale: init, setAxisSpec, setFields, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

LogScale.SETBASE(VALUE)

Specifies the base of the logarithmic scale. The default is 10.

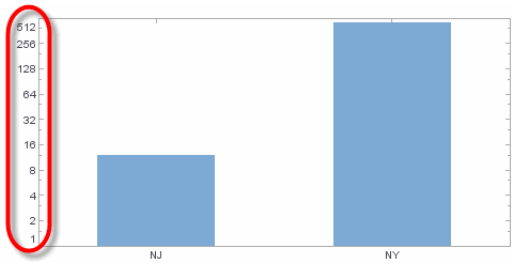
Parameter

value	a number specifying the logarithm base

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ",12], ["NY",450]];
graph = new EGraph();
var qscale = new LogScale("Quantity");
var elem = new IntervalElement("State", "Quantity")
qscale.setBase(2);
graph.setScale("Quantity", qscale);
graph.addElement(elem);
```



Inherits from

LinearScale: setIncrement, setMax, setMin, setMinorIncrement, setReversed, setScaleRange
Scale: init, setAxisSpec, setFields, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.6.11 PowerScale

The PowerScale object contains a scale that maps values to physical attributes by raising them to a specified exponent. To create a PowerScale object, call the PowerScale constructor with the fields for which the scale should be generated.

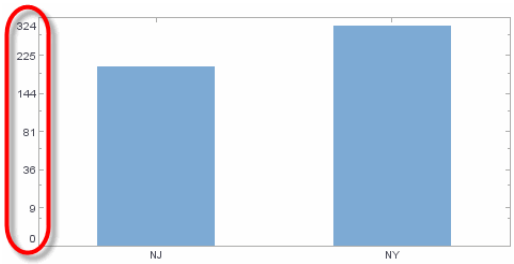
```
var qscale = new PowerScale('Last Year', 'This Year');
```

You can pass the names of the fields (e.g., ‘Last Year’, ‘This Year’) for which the scale should be generated to the constructor, or specify these later using the inherited *Scale.setFields(field)* property.

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ",200], ["NY",300]];
graph = new EGraph();
var qscale = new PowerScale();
qscale.setFields(["Quantity"])
var elem = new IntervalElement("State", "Quantity")
qscale.setExponent(0.5);
graph.setScale("Quantity", qscale);
graph.addElement(elem);
```



Full Name

`inetsoft.graph.scale.PowerScale`

PowerScale: setExponent

Inherits from

LinearScale: setIncrement, setMax, setMin, setMinorIncrement, setReversed, setScaleRange
Scale: init, setAxisSpec, setFields, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

PowerScale.SETEXPONENT(VALUE)

Specifies the exponent of the scaling. The axis position of value `x` is given by `x^exponent`.

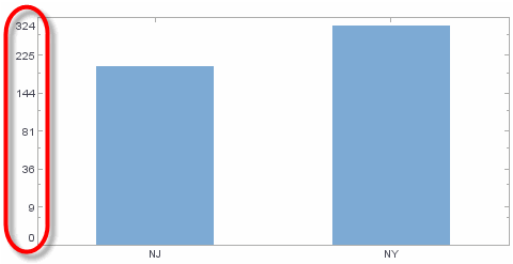
Parameter

value a number specifying the scaling exponent

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ",200], ["NY",300]];
graph = new EGraph();
var qscale = new PowerScale();
qscale.setFields(["Quantity"])
var elem = new IntervalElement("State", "Quantity")
qscale.setExponent(0.5);
graph.setScale("Quantity", qscale);
graph.addElement(elem);
```



Inherits from
LinearScale: setIncrement, setMax, setMin, setMinorIncrement, setReversed, setScaleRange
Scale: init, setAxisSpec, setFields, setScaleOption

See Also
DataSet, to use a Data Block or query as the *dataset* source.

CR.6.12 TimeScale

The TimeScale object contains a time scale, i.e., a scale that linearly maps date and time data values to physical attributes. To create a TimeScale object, call the TimeScale constructor with the fields for which the scale should be generated.

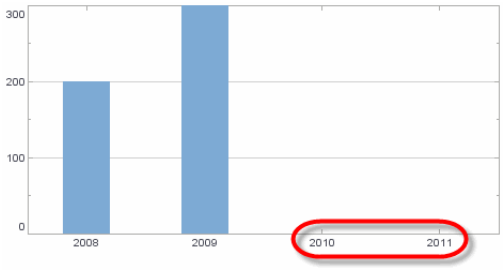
```
var qscale = new TimeScale('Date');
```

You can pass the names of the fields (e.g., 'Date') for which the scale should be generated to the constructor, or specify these later using the inherited *Scale.setFields(field)* property.

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
var date1 = new Date();
var date2 = new Date();
var maxDate = new Date();
date1.setFullYear(2008,0,1);
date2.setFullYear(2009,0,1);
maxDate.setFullYear(2011,0,1);
dataset = [["Date", "Quantity"], [date1,200], [date2,300]];
graph = new EGraph();
var elem = new IntervalElement("Date","Quantity")
var tscale = new TimeScale();
tscale.setFields(["Date"]);
tscale.setMax(maxDate);
graph.setScale("Date", tscale);
graph.addElement(elem);
```



Full Name

`inetsoft.graph.scale.TimeScale`

TimeScale: setIncrement, setMax, setMin, setType

Inherits from

Scale: init, setAxisSpec, setFields, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

TimeScale.SETINCREMENT(VALUE)

Specifies an integer increment at which to place labels on the TimeScale axis. The integer `value` designates the increment in terms of the prevailing time unit, e.g., Weeks, Months, Quarters, etc.

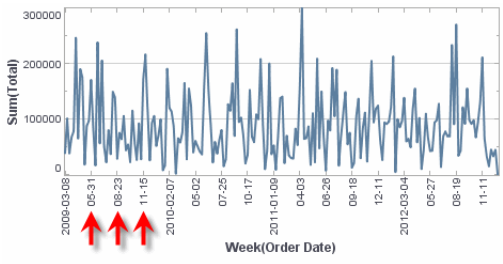
Parameter

`value` `an integer`

`Note: Script that modifies 'graph' should be placed at the element level. See` *`Adding Element-Level Script`* `in` *`Report Scripting`* `and` *`Adding Component Script`* `in` *`Dashboard Scripting`* `for more information.`

Example (Report or Viewsheet)

```
graph.getScale("Order Date").setIncrement(12)
```



Inherits from

Scale: init, setAxisSpec, setFields, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

TimeScale.SETMAX(VALUE)

Specifies the latest date on the scale.

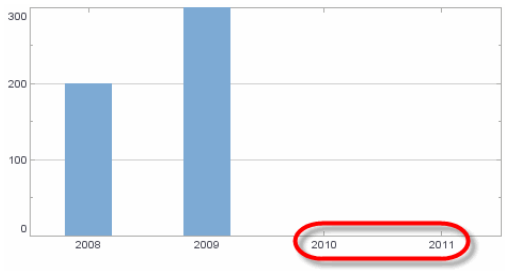
Parameter

value a Date object

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
var date1 = new Date();
var date2 = new Date();
var maxDate = new Date();
date1.setFullYear(2008,0,1);
date2.setFullYear(2009,0,1);
maxDate.setFullYear(2011,0,1);
dataset = [["Date", "Quantity"], [date1,200], [date2,300]];
graph = new EGraph();
var elem = new IntervalElement("Date","Quantity")
var tscale = new TimeScale();
tscale.setFields(["Date"]);
tscale.setMax(maxDate);
graph.setScale("Date", tscale);
graph.addElement(elem);
```



If you set the maximum scale value smaller than the largest data value (so as to crop the data), you should also set `elem.setInPlot(false)`. Otherwise, the chart will force the data to appear in its entirety, resulting in a partially unlabeled axis.

Inherits from

Scale: init, setAxisSpec, setFields, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

TimeScale.SETMIN(VALUE)

Specifies the earliest date on the scale.

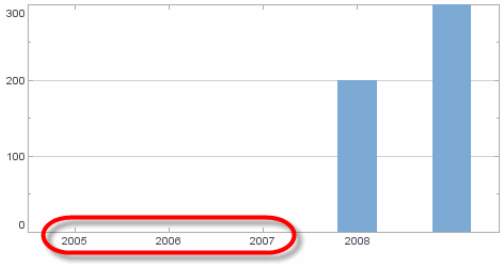
Parameter

value a Date object

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
var date1 = new Date();
var date2 = new Date();
var minDate = new Date();
date1.setFullYear(2008,0,1);
date2.setFullYear(2009,0,1);
minDate.setFullYear(2005,0,1);
dataset = [["Date", "Quantity"], [date1,200], [date2,300]];
graph = new EGraph();
var elem = new IntervalElement("Date","Quantity")
var tscale = new TimeScale("Date");
tscale.setMin(minDate);
graph.setScale("Date", tscale);
graph.addElement(elem);
```



Inherits from

Scale: init, setAxisSpec, setFields, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

TimeScale.SETTYPE(TYPE)

Specifies the placement of tick marks on a TimeScale axis. (This is analogous to the increment setting for a *LinearScale* axis.)

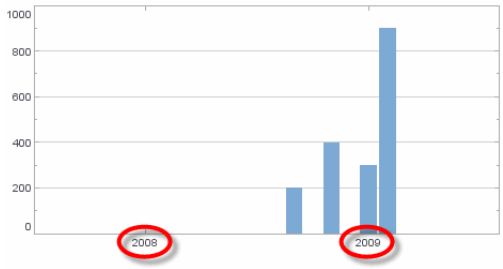
Parameter

type an axis increment specifier:
- TimeScale.DAY
- TimeScale.HOUR
- TimeScale.MINUTE
- TimeScale.MONTH
- TimeScale.QUARTER
- TimeScale.SECOND
- TimeScale.WEEK
- TimeScale.YEAR

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["Date", "Quantity"], [new Date(2008,8,1),200],
[new Date(2008,10,1),400], [new Date(2009,0,1),300], [new
Date(2009,1,1),900]];
graph = new EGraph();
var elem = new IntervalElement("Date","Quantity")
var tscale = new TimeScale("Date");
tscale.setType(TimeScale.YEAR)
graph.setScale("Date", tscale);
graph.addElement(elem);
```



Inherits from

Scale: init, setAxisSpec, setFields, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.6.13 CategoricalScale

The CategoricalScale object contains a nominal scale, i.e., a scale that logically maps nominal values to physical attributes. To create a CategoricalScale object, call the CategoricalScale constructor with the fields for which the scale should be generated.

```
var qscale = new CategoricalScale('State');
```

You can pass the names of the fields (e.g., 'State') for which the scale should be generated to the constructor, or specify these later using the inherited *Scale.setFields(field)* property.

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State","Quantity"], ["NJ",200], ["NY",300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var sscale = new CategoricalScale("State");
var qscale = new LinearScale("Quantity");
var coord = new RectCoord(sscale, qscale);
coord.transpose();
graph.setCoordinate(coord);
graph.addElement(elem);
```

Full Name

```
inetsoft.graph.scale.CategoricalScale
```

CategoricalScale: setValues, setFill

Inherits from

Scale: init, setAxisSpec, setFields, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

CategoricalScale.SETFILL(BOOLEAN)

Specifies that scale boundaries should be set equal to the extreme data values (leaving no gap at the axis edges).

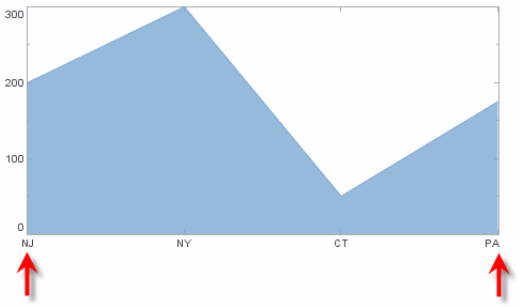
Parameter

Boolean	true, to fill axis to edges false, to leave gap at edges

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ", 200], ["NY", 300],
["CT", 50], ["PA", 175]];
graph = new EGraph();
var elem = new AreaElement("State", "Quantity");
var sscale = new CategoricalScale("State");
sscale.setFill(true)
var qscale = new LinearScale("Quantity");
var coord = new RectCoord(sscale, qscale);
graph.setCoordinate(coord);
graph.addElement(elem);
```



Inherits from

Scale: init, setAxisSpec, setFields, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

CategoricalScale.SETVALUES(VALUE)

Specifies the categorical values in the scale, and their order.

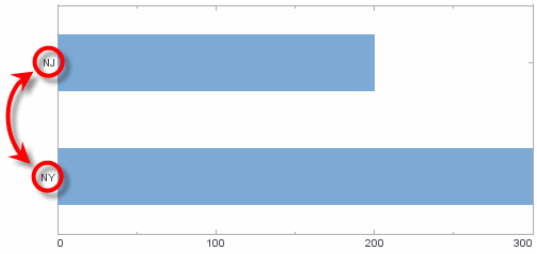
Parameter

value Array of String

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ", 200], ["NY", 300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var sscale = new CategoricalScale("State");
var qscale = new LinearScale("Quantity");
var coord = new RectCoord(sscale, qscale);
sscale.setValues(["NY","NJ"]);
coord.transpose();
graph.setCoordinate(coord);
graph.addElement(elem);
```



Inherits from

Scale: init, setAxisSpec, setFields, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.6.14 ScaleRange

The ScaleRange object contains the calculation strategy for finding the scale range.

Full Name

inetsoft.graph.scale.ScaleRange

ScaleRange: setAbsoluteValue

Created by

StackRange
LinearRange

ScaleRange.SETABSOLUTEVALUE(BOOLEAN)

Specifies whether negative quantities should be represented against the positive axis or against the negative axis (default).

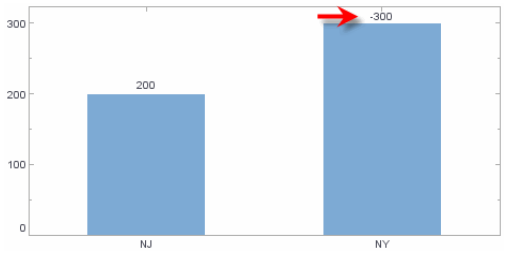
Parameter

boolean	true: show neg values on pos axis false: show neg values on neg axis

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ",200], ["NY",-300]];
graph = new EGraph();
var qscale = new LinearScale("Quantity");
var elem = new IntervalElement("State", "Quantity");
var frame = new DefaultTextFrame();
frame.setField("Quantity")
elem.setTextFrame(frame)
var range = new LinearRange();
range.setAbsoluteValue(true);
qscale.setScaleRange(range);
graph.setScale("Quantity", qscale);
graph.addElement(elem);
```



See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.6.15 StackRange

The StackRange object computes the range by "stacking" the data values. To create a StackRange object, call the StackRange constructor.

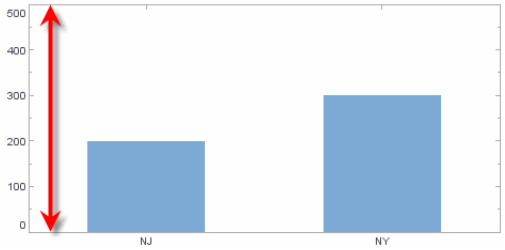
```
var range = new StackRange();
```

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ",200], ["NY",300]];
graph = new EGraph();
var qscale = new LinearScale("Quantity");
```

```
var elem = new IntervalElement("State", "Quantity")
qscale.setScaleRange(new StackRange());  // adds 200+300
graph.setScale("Quantity", qscale);
graph.addElement(elem);
```



Full Name

`inetsoft.graph.scale.StackRange`

StackRange: setGroupField, setStackNegative

Inherits from

ScaleRange: setAbsoluteValue

See Also

DataSet, to use a Data Block or query as the *dataset* source.

StackRange.SETGROUPFIELD(VALUE)

Determines the scale range from the stacked values of largest single group, based on the specified grouping field.

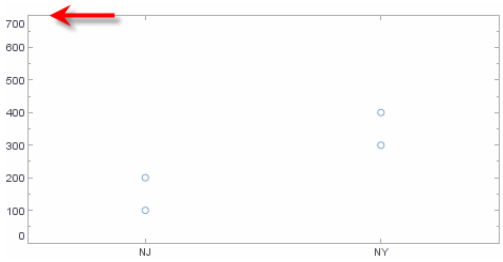
Parameter

value the field name, a String

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State",
"Quantity"],["NJ",200],["NJ",100],["NY",400],["NY",300]];
graph = new EGraph();
var qscale = new LinearScale("Quantity");
var elem = new PointElement("State", "Quantity")
range = new StackRange();
range.setGroupField("State"); // max of 200+100, 400+300
qscale.setScaleRange(range);
graph.setScale("Quantity", qscale);
graph.addElement(elem);
```



Inherits from
ScaleRange: setAbsoluteValue

See Also
DataSet, to use a Data Block or query as the *dataset* source.

StackRange.SETSTACKNEGATIVE(BOOLEAN)

Specifies whether the negative scale range is determine by independently stacking the negative values (default), or whether stacking is not applied to negative values in computing the range.

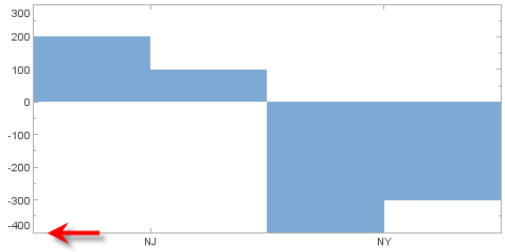
Parameter

boolean	`true: stack negative values to get range` `false: do not stack negative values`

`Note: Script that modifies 'graph' should be placed at the element level. See` *`Adding Element-Level Script`* `in` *`Report Scripting`* `and` *`Adding Component Script`* `in` *`Dashboard Scripting`* `for more information.`

Example (Report or Viewsheet)

```
dataset = [["State",
"Quantity"],["NJ",200],["NJ",100],["NY",-300],["NY",-400]];
graph = new EGraph();
var qscale = new LinearScale("Quantity");
var elem = new IntervalElement("State", "Quantity")
range = new StackRange();
range.setStackNegative(false);
qscale.setScaleRange(range);
graph.setScale("Quantity", qscale);
graph.addElement(elem);
```



Inherits from
ScaleRange: setAbsoluteValue

See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.6.16 LinearRange

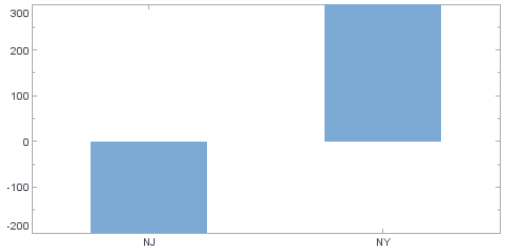
The LinearRange object computes the range by using the minimum and maximum data values. To create a LinearRange object, call the LinearRange constructor.

```
var range = new LinearRange();
```

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"],["NJ",-200],["NY",300]];
graph = new EGraph();
var qscale = new LinearScale("Quantity");
var elem = new IntervalElement("State", "Quantity")
range = new LinearRange();
qscale.setScaleRange(range);
graph.setScale("Quantity", qscale);
graph.addElement(elem);
```



Full Name

```
inetsoft.graph.scale.LinearRange
```

Inherits from

ScaleRange: setAbsoluteValue

See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.7 Chart Aesthetics

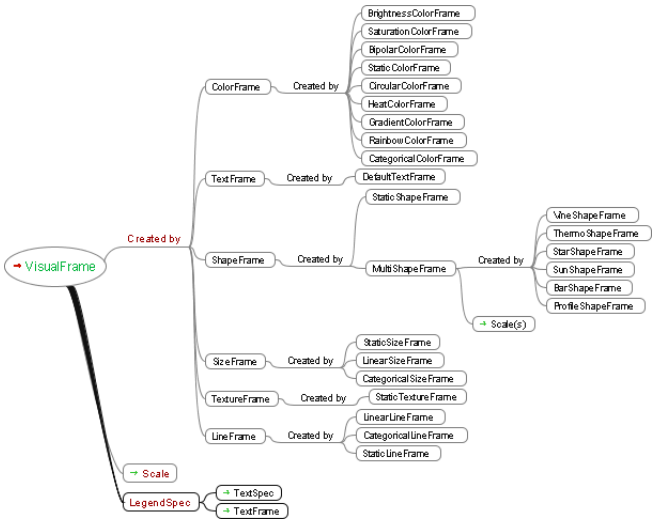
This section discusses the *VisualFrame* objects that can be added to chart elements to introduce visual style. *VisualFrame* objects allow you to represent additional data dimensions by using the physical attributes of chart elements, or to apply a fixed (static) visual style.

CR.7.1 VisualFrame

The VisualFrame object contains common properties for all aesthetic frames.

Full Name

`inetsoft.graph.aesthetic.VisualFrame`



VisualFrame: setField, setLegendSpec, setScale, setScaleOption

Created by
ColorFrame
SizeFrame
ShapeFrame
TextFrame
LineFrame
TextureFrame

VisualFrame.SETFIELD(FIELD)

Specifies the field (column) associated with this frame.

Parameter

field name of the column (String)

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ",200], ["NY",300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var frame = new BrightnessColorFrame();
frame.setField("Quantity");
```

```
frame.setColor(java.awt.Color(0xff0000));
elem.setColorFrame(frame);
graph.addElement(elem);
```

See Also

DataSet, to use a Data Block or query as the *dataset* source.

VisualFrame.SETLEGENDSPEC(SPEC)

Specifies the formatting for the legend generated for the scale.

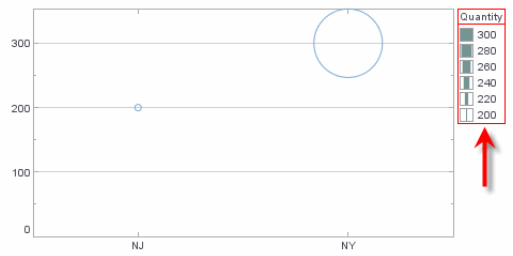
Parameter

spec a *LegendSpec* object

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ",200], ["NY",300]];
graph = new EGraph();
var elem = new PointElement("State", "Quantity");
var frame = new LinearSizeFrame();
var spec = new LegendSpec();
spec.setBorderColor(java.awt.Color(0xff0000));
frame.setField("Quantity");
frame.setLegendSpec(spec);
elem.setSizeFrame(frame);
graph.addElement(elem);
```



See Also

DataSet, to use a Data Block or query as the *dataset* source.

VisualFrame.SETSCALE(SCALE)

Specifies the scale associated with this frame.

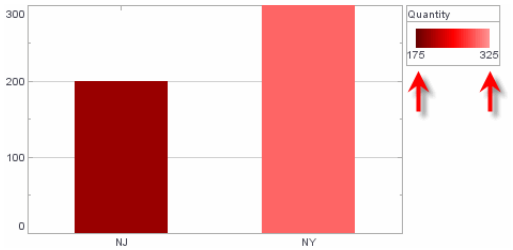
Parameter

scale a *Scale* object

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ",200], ["NY",300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var frame = new BrightnessColorFrame();
frame.setField("Quantity");
var scale = new LinearScale();
scale.setMax(325);
scale.setMin(175);
frame.setScale(scale);
frame.setColor(java.awt.Color(0xff0000));
elem.setColorFrame(frame);
graph.addElement(elem);
```



See Also

DataSet, to use a Data Block or query as the *dataset* source.

VisualFrame.SETSCALEOPTION(VALUE)

Specifies a scaling option for the default scaling. The `Scale.TICKS` and `Scale.ZERO` options determine the maximum and minimum values that are used to calculate the scale range.

The `Scale.TICKS` option uses the maximum and minimum tick values (i.e., rounded numbers) rather than the maximum and minimum data values. The `Scale.ZERO` option uses zero as the minimum rather than the minimum data value (if positive). To combine multiple options, use the pipe ("or") operator:

```
frame.setScaleOption(Scale.ZERO | Scale.TICKS);
```

Parameter

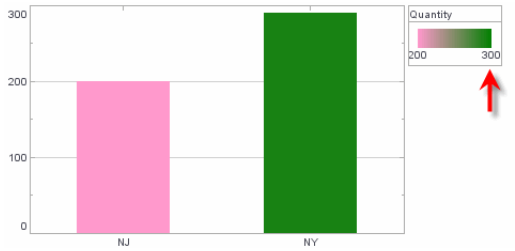
value	`Scale.TICKS` `Scale.ZERO`

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ",200], ["NY",290]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var frame = new GradientColorFrame();
frame.setField("Quantity");
```

```
frame.setScaleOption(Scale.TICKS);
elem.setColorFrame(frame);
graph.addElement(elem);
```



See Also

DataSet, to use a Data Block or query as the *dataset* source.

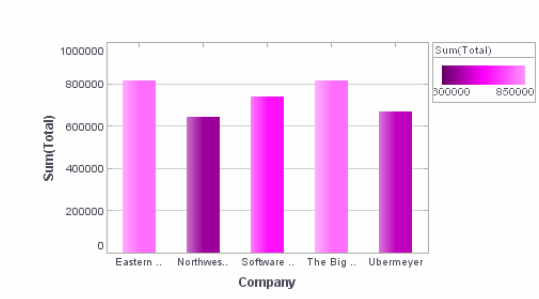
CR.7.2 ColorFrame

A ColorFrame object contains the color treatment for visual chart elements. You can use a ColorFrame object to represent data dimensions with color (color coding), or to apply a fixed (static) color style.

Note: Report script that modifies 'bindingInfo' should be placed in the onLoad handler.

Example (Report)

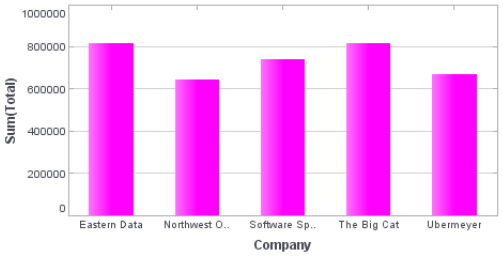
```
Chart1.bindingInfo.setColorField("Total",Chart.NUMBER);
Chart1.bindingInfo.colorFrame = new BrightnessColorFrame;
Chart1.bindingInfo.colorFrame.color = 0xFF00FF;
```



Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Viewsheet or Report)

```
var frame = new StaticColorFrame;
frame.color = java.awt.Color(0xFF00FF);
var elem = graph.getElement(0); // bar elements
elem.colorFrame = frame;
```



Full Name

```
inetsoft.graph.aesthetic.ColorFrame
```

Created by

BrightnessColorFrame
SaturationColorFrame
BipolarColorFrame
StaticColorFrame
CircularColorFrame
GradientColorFrame
HeatColorFrame
RainbowColorFrame
CategoricalColorFrame

Inherits from

VisualFrame: setField, setLegendSpec, setScale, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.7.3 BrightnessColorFrame

The BrightnessColorFrame object contains a continuous color frame that returns varying brightnesses of the specified color. To create a BrightnessColorFrame object, call the BrightnessColorFrame constructor.

```
var frame = new BrightnessColorFrame('Quantity');
```

You can pass a field name to the constructor (e.g., “Quantity”), or specify this later using the inherited *VisualFrame.setField(field)* property.

Full Name

```
inetsoft.graph.aesthetic.BrightnessColorFrame
```

Inherits from

VisualFrame: setField, setLegendSpec, setScale, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

BrightnessColorFrame: setColor

BrightnessColorFrame.SETCOLOR(VALUE)

Specifies the color whose brightness is varied. The data values in the associated column (specified by the inherited *VisualFrame.setField(field)* property) are mapped to a spectrum of brightnesses of the specified color.

Value (Property Syntax)

java.awt.Color	e.g., java.awt.Color.BLUE
number (hex)	e.g., 0xFF0000
string (color name)	e.g., 'red'
array [r,g,b]	e.g., [255,0,0]
JSON {r:#,g:#,b:#}	e.g., {r:255,g:0,b:0}

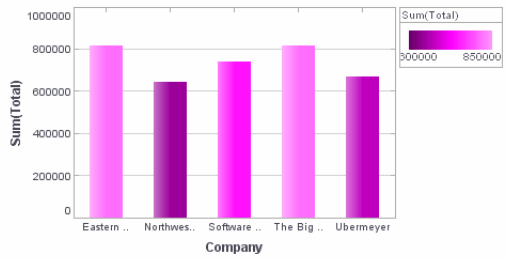
Value (Function Syntax)

value a java.awt.Color object

Note: Report script that modifies 'bindingInfo' should be placed in the onLoad handler.

Example (Report)

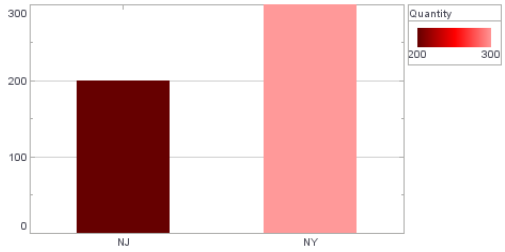
```
Chart1.bindingInfo.setColorField("Total",Chart.NUMBER);
Chart1.bindingInfo.colorFrame = new BrightnessColorFrame;
Chart1.bindingInfo.colorFrame.color = 0xFF00FF;
```



Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Viewsheet or Report)

```
dataset = [["State", "Quantity"], ["NJ",200], ["NY",300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var frame = new BrightnessColorFrame();
frame.setField("Quantity")
frame.setColor(java.awt.Color(0xff0000));
elem.setColorFrame(frame);
graph.addElement(elem);
```



Inherits from
VisualFrame: setField, setLegendSpec, setScale, setScaleOption

See Also
DataSet, to use a Data Block or query as the *dataset* source.

CR.7.4 SaturationColorFrame

The SaturationColorFrame object contains a continuous color frame that returns varying saturations of the specified color. To create a SaturationColorFrame object, call the SaturationColorFrame constructor.

```
var frame = new SaturationColorFrame('Quantity');
```

You can pass a field name to the constructor (e.g., “Quantity”), or specify this later using the inherited *VisualFrame.setField(field)* property.

Full Name
`inetsoft.graph.aesthetic.SaturationColorFrame`

Inherits from
VisualFrame: setField, setLegendSpec, setScale, setScaleOption

See Also
DataSet, to use a Data Block or query as the *dataset* source.

SaturationColorFrame: setColor

SaturationColorFrame.SETCOLOR(VALUE)

Specifies the color whose saturation is varied. The data values in the associated column (specified by the inherited *VisualFrame.setField(field)* property) are mapped to a spectrum of saturations of the specified color.

Type

`java.awt.Color`	`e.g., java.awt.Color.BLUE`
`number (hex)`	`e.g., 0xFF0000`
`string (color name)`	`e.g., 'red'`
`array [r,g,b]`	`e.g., [255,0,0]`
`JSON {r:#,g:#,b:#}`	`e.g., {r:255,g:0,b:0}`

Value (Function Syntax)

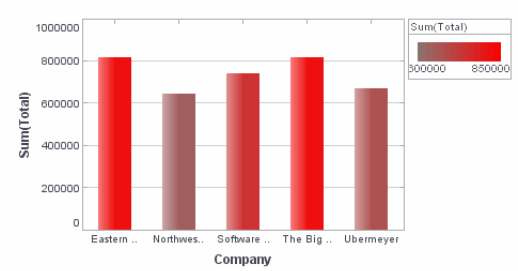
`value` a `java.awt.Color` object

Note: Report script that modifies 'bindingInfo' should be placed in the onLoad handler.

Example (Report)

Bind a bar-type chart to the sample 'All Sales' query, with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the *onLoad Handler*.

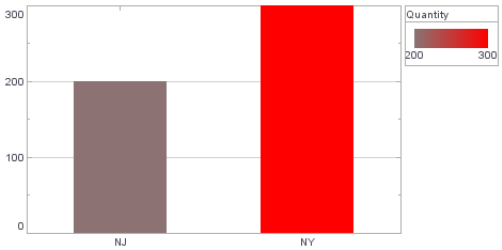
```
Chart1.bindingInfo.setColorField("Total",Chart.NUMBER);
Chart1.bindingInfo.colorFrame = new SaturationColorFrame;
Chart1.bindingInfo.colorFrame.color = 0xFF0000;
```



Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Viewsheet or Report)

```
dataset = [["State", "Quantity"], ["NJ",200], ["NY",300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var frame = new SaturationColorFrame();
frame.setField("Quantity");
frame.setColor(java.awt.Color(0xff0000));
elem.setColorFrame(frame);
graph.addElement(elem);
```



Inherits from

VisualFrame: setField, setLegendSpec, setScale, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.7.5 BipolarColorFrame

The BipolarColorFrame object contains a continuous color frame that returns gradations between two colors. To create a BipolarColorFrame object, call the BipolarColorFrame constructor.

```
var frame = new BipolarColorFrame('Quantity');
```

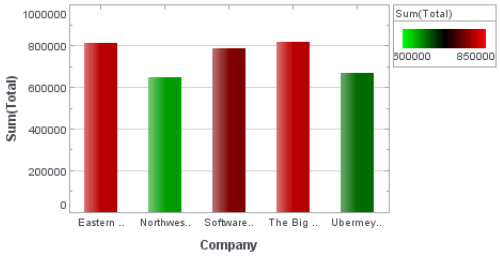
You can pass a field name to the constructor (e.g., “Quantity”), or specify this later using the inherited *VisualFrame.setField(field)* property.

Note: Report script that modifies 'bindingInfo' should be placed in the onLoad handler.

Example (Report)

Bind a bar-type chart to the sample ‘All Sales’ query, with ‘Company’ (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the *onLoad Handler*.

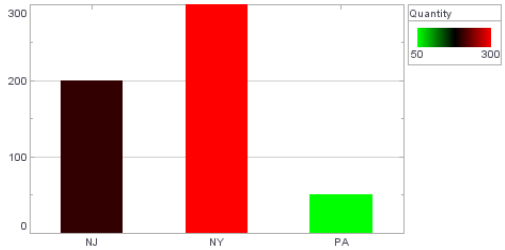
```
Chart1.bindingInfo.setColorField("Total",Chart.NUMBER);
Chart1.bindingInfo.colorFrame = new BipolarColorFrame;
```



Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State",
"Quantity"],["NJ",200],["NY",300],["PA",50]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var frame = new BipolarColorFrame();
frame.setField("Quantity");
elem.setColorFrame(frame);
graph.addElement(elem);
```



Full Name

```
inetsoft.graph.aesthetic.BipolarColorFrame
```

Inherits from

VisualFrame: setField, setLegendSpec, setScale, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.7.6 StaticColorFrame

The StaticColorFrame object contains a color frame defined by explicit color data in the *VisualFrame.setField(field)* column, or by the fixed color in *StaticColorFrame.setColor(value)*. To create a StaticColorFrame object, call the StaticColorFrame constructor.

```
var frame = new StaticColorFrame();
```

You can pass a color value directly to the constructor, e.g.,

```
var frame = new StaticColorFrame(java.awt.Color(0xFF00FF));
```

or specify it later using the in *StaticColorFrame.setColor(value)* property.

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State","Quantity","Color"],
["NJ",200,0xff0000], ["NY",300,0xff00ff]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var frame = new StaticColorFrame();
frame.setField("Color");
elem.setColorFrame(frame);
graph.addElement(elem);
```

Full Name

```
inetsoft.graph.aesthetic.StaticColorFrame
```

Inherits from

VisualFrame: setField, setLegendSpec, setScale, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

StaticColorFrame: setColor, setNegativeColor

StaticColorFrame.SETCOLOR(VALUE)

Specifies the static color to be used for positive field values. The static color can also be passed as an input to the constructor. If the data column

specified by the inherited *VisualFrame.setField(field)* property contains (positive) numbers or colors, these data values are used instead of *StaticColorFrame.setColor()*.

Value (Property Syntax)

```
java.awt.Color          e.g., java.awt.Color.BLUE
number (hex)            e.g., 0xFF0000
string (color name)     e.g., 'red'
array [r,g,b]           e.g., [255,0,0]
JSON {r:#,g:#,b:#}      e.g., {r:255,g:0,b:0}
```

Value (Function Syntax)

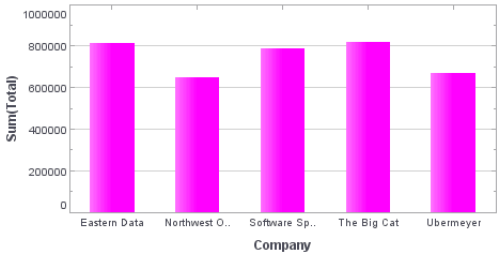
value a java.awt.Color object

Note: Report script that modifies 'bindingInfo' should be placed in the onLoad handler.

Example (Report)

Bind a bar-type chart to the sample 'All Sales' query, with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the *onLoad Handler*.

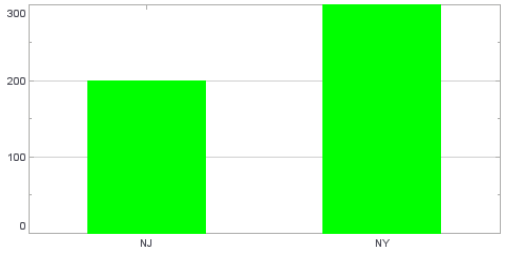
```
Chart1.bindingInfo.setColorField("Company",Chart.STRING);
Chart1.bindingInfo.colorFrame = new StaticColorFrame;
Chart1.bindingInfo.colorFrame.color = 0xFF00FF;
```



Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Viewsheet or Report)

```
dataset = [["State", "Quantity"], ["NJ",200], ["NY",300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var frame = new StaticColorFrame();
frame.setColor(java.awt.Color(0x00ff00));
elem.setColorFrame(frame);
graph.addElement(elem);
```



Inherits from
VisualFrame: setField, setLegendSpec, setScale, setScaleOption

See Also
DataSet, to use a Data Block or query as the *dataset* source.

StaticColorFrame.SETNEGATIVECOLOR(VALUE)

Specifies the static color to be used for negative field values. If a value is specified for *StaticColorFrame.setNegativeColor()*, then *StaticColorFrame.setColor()* defines the color of positive values, and *StaticColorFrame.setNegativeColor()* defines the color of negative values. In this case, the inherited *VisualFrame.setField(field)* property is not used.

Value (Property Syntax)

```
java.awt.Color          e.g., java.awt.Color.BLUE
number (hex)            e.g., 0xFF0000
string (color name)     e.g., 'red'
array [r,g,b]           e.g., [255,0,0]
JSON {r:#,g:#,b:#}      e.g., {r:255,g:0,b:0}
```

Value (Function Syntax)

```
value          a java.awt.Color object
```

```
Note: Report script that modifies 'bindingInfo' should be placed in
the onLoad handler.
```

Example (Report)

In this example, you will set the color of negative values for a bar chart. Follow the steps below:

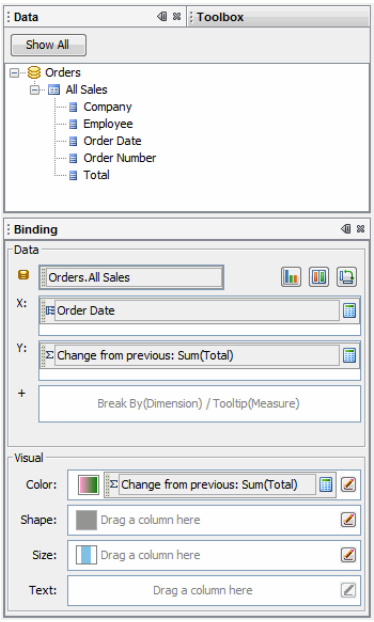
1. Create a new report, and add a Chart component to the report.
2. Click on the chart to open the Chart Editor.
3. In the **Data** tab of the Chart Editor, expand the ‘All Sales’ query.
4. Drag the ‘Order Date’ field from the **Data** tab to the ‘X’ region.
5. Press the ‘Edit Dimension’ button next to the ‘X’ region. Select ‘Quarter’ from the ‘Level’ menu, and press ‘Apply’.

6. Drag the 'Total' field from the **Data** tab to the 'Y' region.

7. Press the 'Edit Measure' button next to the 'Y' region. Select 'Change from Previous' from the 'Calculate' menu, and press 'Apply'.

8. Drag the 'Total' field from the **Data** tab to the 'Color' region.

9. Press the 'Edit Measure' button next to the 'Color' region. Select 'Change from Previous' from the 'Calculate' menu, and press 'Apply'.

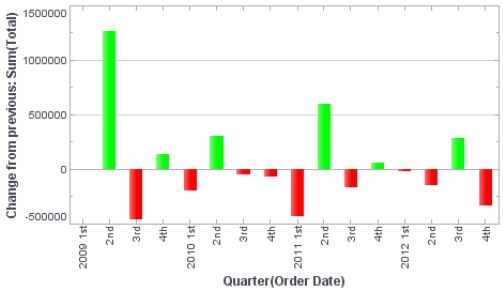


10. Add the following script in the onLoad handler.

```
Chart1.bindingInfo.colorFrame = new StaticColorFrame;
Chart1.bindingInfo.colorFrame.color = 0x00FF00;
Chart1.bindingInfo.colorFrame.negativeColor = 0xFF0000;
```

Note: You can use *highlighting* to achieve the same effect without script.

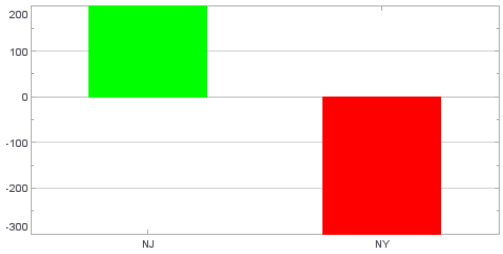
11. Preview the report, and observe that negative-valued bars are displayed in the specified static color.



Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Viewsheet or Report)

```
dataset = [["State", "Quantity"], ["NJ",200], ["NY",-300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var frame = new StaticColorFrame();
frame.setField("Quantity");
frame.setColor(java.awt.Color(0x00ff00));
frame.setNegativeColor(java.awt.Color(0xff0000));
elem.setColorFrame(frame);
graph.addElement(elem);
```



Inherits from

VisualFrame: setField, setLegendSpec, setScale, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.7.7 CircularColorFrame

The CircularColorFrame object contains a continuous color frame that returns gradations from the full spectrum. To create a CircularColorFrame object, call the CircularColorFrame constructor.

```
var frame = new CircularColorFrame('Quantity');
```

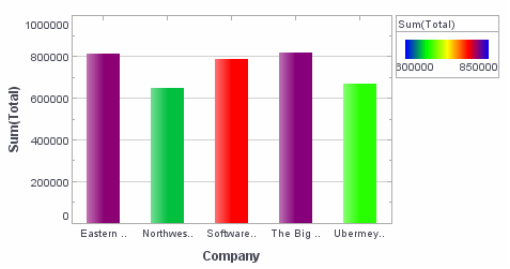
You can pass the name of a field (e.g., 'Quantity') to the constructor, or specify this later using the inherited *VisualFrame.setField(field)* property.

Note: Report script that modifies 'bindingInfo' should be placed in the onLoad handler.

Example (Report)

Bind a bar-type chart to the sample 'All Sales' query, with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

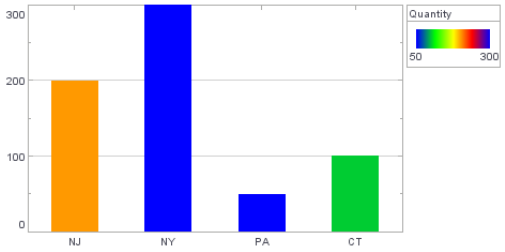
```
Chart1.bindingInfo.setColorField("Total",Chart.NUMBER);
Chart1.bindingInfo.colorFrame = new CircularColorFrame;
```



Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Viewsheet or Report)

```
dataset = [["State",
"Quantity"],["NJ",200],["NY",300],["PA",50],["CT",100]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var frame = new CircularColorFrame();
frame.setField("Quantity");
elem.setColorFrame(frame);
graph.addElement(elem);
```



Full Name

`inetsoft.graph.aesthetic.CircularColorFrame`

Inherits from

VisualFrame: setField, setLegendSpec, setScale, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.7.8 GradientColorFrame

The GradientColorFrame object contains a continuous color frame that returns gradations between two colors. To create a GradientColorFrame object, call the GradientColorFrame constructor.

```
var frame = new GradientColorFrame();
```

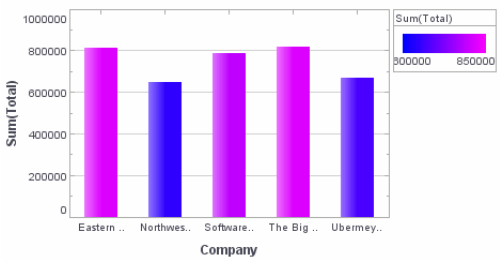
You can pass the name of a field (e.g., 'Quantity') to the constructor, or specify this later using the inherited *VisualFrame.setField(field)* property.

Note: Report script that modifies 'bindingInfo' should be placed in the onLoad handler.

Example (Report)

Bind a bar-type chart to the sample 'All Sales' query, with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

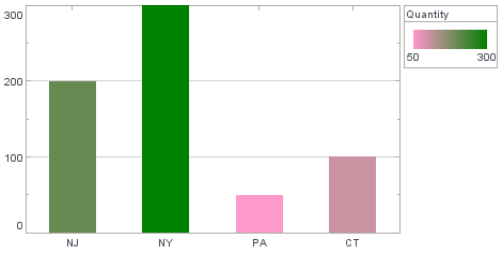
```
Chart1.bindingInfo.setColorField("Total",Chart.NUMBER);
Chart1.bindingInfo.colorFrame = new GradientColorFrame;
Chart1.bindingInfo.colorFrame.fromColor = 0x0000FF;
Chart1.bindingInfo.colorFrame.toColor = 0xFF00FF;
```



Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State",
"Quantity"],["NJ",200],["NY",300],["PA",50],["CT",100]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var frame = new GradientColorFrame();
frame.setField("Quantity");
elem.setColorFrame(frame);
graph.addElement(elem);
```



Full Name

inetsoft.graph.aesthetic.GradientColorFrame

Inherits from

VisualFrame: setField, setLegendSpec, setScale, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

GradientColorFrame: setFromColor, setToColor

GradientColorFrame.SETFROMCOLOR(VALUE)

Specifies the starting color (for lowest value) of the gradient.

Value (Property Syntax)

java.awt.Color	e.g., java.awt.Color.BLUE
number (hex)	e.g., 0xFF0000
string (color name)	e.g., 'red'
array [r,g,b]	e.g., [255,0,0]
JSON {r:#,g:#,b:#}	e.g., {r:255,g:0,b:0}

Value (Function Syntax)

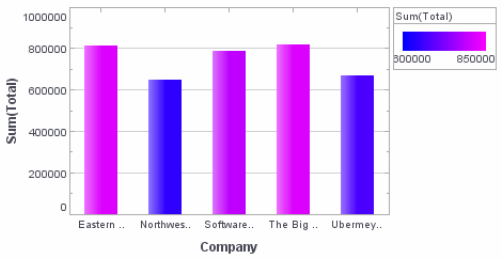
value a java.awt.Color object

Note: Report script that modifies 'bindingInfo' should be placed in the onLoad handler.

Example (Report)

Bind a bar-type chart to the sample 'All Sales' query, with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

```
Chart1.bindingInfo.setColorField("Total",Chart.NUMBER);
Chart1.bindingInfo.colorFrame = new GradientColorFrame;
Chart1.bindingInfo.colorFrame.fromColor = 0x0000FF;
Chart1.bindingInfo.colorFrame.toColor = 0xFF00FF;
```

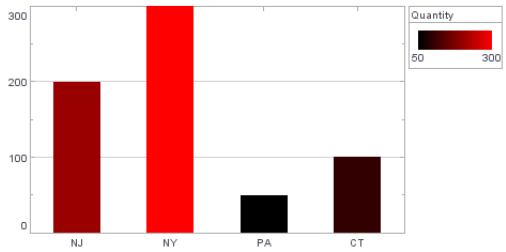


Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State",
"Quantity"],["NJ",200],["NY",300],["PA",50],["CT",100]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var frame = new GradientColorFrame();
frame.setFromColor(java.awt.Color(0x000000));
frame.setToColor(java.awt.Color(0xff0000));
```

```
frame.setField("Quantity");
elem.setColorFrame(frame);
graph.addElement(elem);
```



Inherits from

VisualFrame: setField, setLegendSpec, setScale, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

GradientColorFrame.SETTOCOLOR(VALUE)

Specifies the ending color (for highest value) of the gradient.

Value (Property Syntax)

java.awt.Color	e.g., java.awt.Color.BLUE
number (hex)	e.g., 0xFF0000
string (color name)	e.g., 'red'
array [r,g,b]	e.g., [255,0,0]
JSON {r:#,g:#,b:#}	e.g., {r:255,g:0,b:0}

Value (Function Syntax)

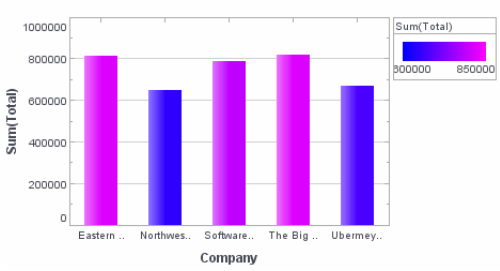
value a java.awt.Color object

Note: Report script that modifies 'bindingInfo' should be placed in the onLoad handler.

Example (Report)

Bind a bar-type chart to the sample ‘All Sales’ query, with ‘Company’ (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

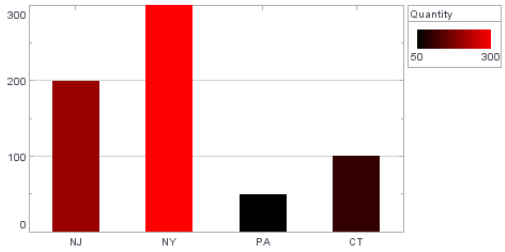
```
Chart1.bindingInfo.setColorField("Total",Chart.NUMBER);
Chart1.bindingInfo.colorFrame = new GradientColorFrame;
Chart1.bindingInfo.colorFrame.fromColor = 0x0000FF;
Chart1.bindingInfo.colorFrame.toColor = 0xFF00FF;
```



Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State",
"Quantity"],["NJ",200],["NY",300],["PA",50],["CT",100]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var frame = new GradientColorFrame();
frame.setFromColor(java.awt.Color(0x000000));
frame.setToColor(java.awt.Color(0xff0000));
frame.setField("Quantity");
elem.setColorFrame(frame);
graph.addElement(elem);
```



Inherits from

VisualFrame: setField, setLegendSpec, setScale, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.7.9 HeatColorFrame

The HeatColorFrame object contains a continuous color frame that returns varying shades of brown. To create a HeatColorFrame object, call the HeatColorFrame constructor.

```
var frame = new HeatColorFrame('Quantity');
```

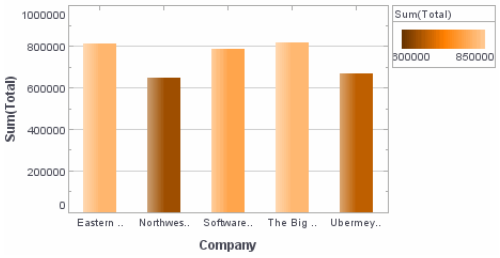
You can pass the name of a field (e.g., 'Quantity') to the constructor, or specify this later using the inherited *VisualFrame.setField(field)* property.

Note: Report script that modifies 'bindingInfo' should be placed in the onLoad handler.

Example (Report)

Bind a bar-type chart to the sample 'All Sales' query, with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

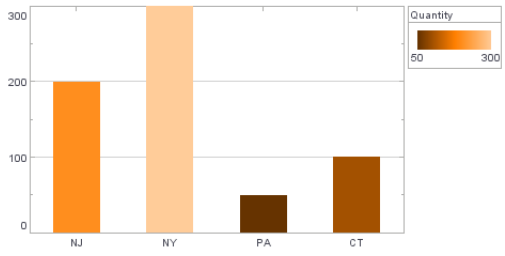
```
Chart1.bindingInfo.setColorField("Total",Chart.NUMBER);
Chart1.bindingInfo.colorFrame = new HeatColorFrame();
```



Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State",
"Quantity"],["NJ",200],["NY",300],["PA",50],["CT",100]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var frame = new HeatColorFrame();
frame.setField("Quantity");
elem.setColorFrame(frame);
graph.addElement(elem);
```



Full Name

```
inetsoft.graph.aesthetic.HeatColorFrame
```

Inherits from

VisualFrame: setField, setLegendSpec, setScale, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.7.10 RainbowColorFrame

The RainbowColorFrame object contains a continuous color frame that returns colors of the rainbow. To create a RainbowColorFrame object, call the RainbowColorFrame constructor.

```
var frame = new RainbowColorFrame('Quantity');
```

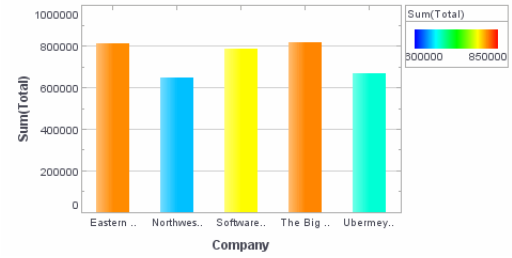
You can pass the name of a field (e.g., 'Quantity') to the constructor, or specify this later using the inherited *VisualFrame.setField(field)* property.

Note: Report script that modifies 'bindingInfo' should be placed in the onLoad handler.

Example (Report)

Bind a bar-type chart to the sample 'All Sales' query, with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

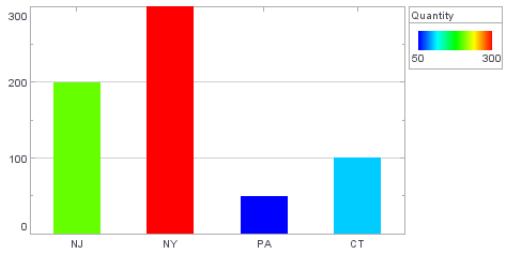
```
Chart1.bindingInfo.setColorField("Total",Chart.NUMBER);
Chart1.bindingInfo.colorFrame = new RainbowColorFrame;
```



Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State",
"Quantity"],["NJ",200],["NY",300],["PA",50],["CT",100]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var frame = new RainbowColorFrame();
frame.setField("Quantity");
elem.setColorFrame(frame);
graph.addElement(elem);
```



Full Name

`inetsoft.graph.aesthetic.RainbowColorFrame`

Inherits from

VisualFrame: setField, setLegendSpec, setScale, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.7.11 RGBCubeColorFrame

The RGBCubeColorFrame object contains a continuous color frame that returns colors drawn from a range of the RGB color cube specified by two boundary colors, `[rgb1,rgb2]`. To create a RGBCubeColorFrame object, call the RGBCubeColorFrame constructor.

```
var frame = new RGBCubeColorFrame('Quantity',[rgb1,rgb2]);
```

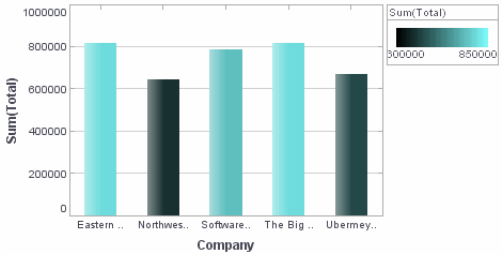
You can pass the name of a field (e.g., 'Quantity') to the constructor, or specify this later using the inherited *VisualFrame.setField(field)* property.

```
Note: Report script that modifies 'bindingInfo' should be placed in
the onLoad handler.
```

Example (Report)

Bind a bar-type chart to the sample 'All Sales' query, with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

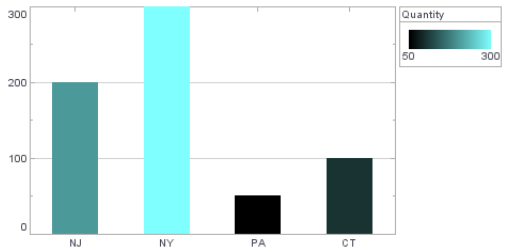
```
Chart1.bindingInfo.setColorField("Total",Chart.NUMBER);
Chart1.bindingInfo.colorFrame = new
RGBCubeColorFrame([[0,0,0],[.5,1,1]]);
```



Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset =
[["State","Quantity"],["NJ",200],["NY",300],["PA",50],["CT"
,100]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var frame = new RGBCubeColorFrame([[0,0,0],[.5,1,1]]);
frame.setField("Quantity");
elem.setColorFrame(frame);
graph.addElement(elem);
```



Full Name

```
inetsoft.graph.aesthetic.RGBCubeColorFrame
```

Inherits from

VisualFrame: setField, setLegendSpec, setScale, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.7.12 CategoricalColorFrame

The CategoricalColorFrame object contains a color frame that returns a distinct color for each unique value in the bound field. To create a CategoricalColorFrame object, call the CategoricalColorFrame constructor.

```
var frame = new CategoricalColorFrame('State');
```

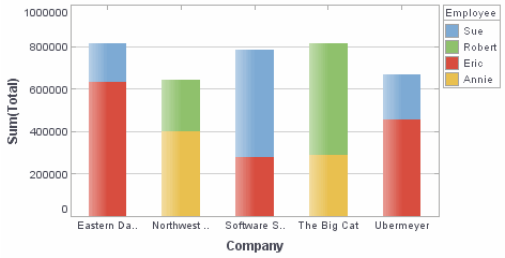
You can pass the name of a field (e.g., 'State') to the constructor, or specify this later using the inherited *VisualFrame.setField(field)* property.

Note: Report script that modifies 'bindingInfo' should be placed in the onLoad handler.

Example (Report)

Bind a bar-type chart to the sample 'All Sales' query, with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

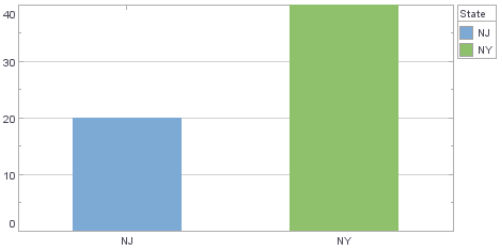
```
Chart1.bindingInfo.setColorField("Employee",Chart.STRING);
Chart1.bindingInfo.colorFrame = new CategoricalColorFrame;
```



Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ",20], ["NY",40]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var frame = new CategoricalColorFrame();
frame.setField("State");
elem.setColorFrame(frame);
graph.addElement(elem);
```



Full Name

`inetsoft.graph.aesthetic.CategoricalColorFrame`

CategoricalColorFrame: getColor, init, setColor

Inherits from

VisualFrame: setField, setLegendSpec, setScale, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

CategoricalColorFrame.GETCOLOR(VAL)

Returns the color assigned to the specified value.

Parameter

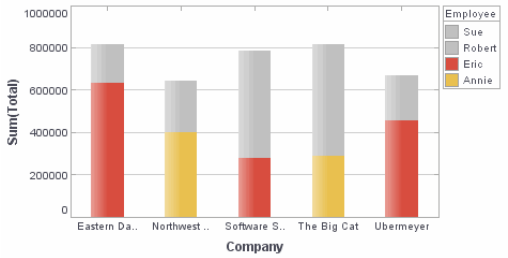
val a data value

Note: Report script that modifies 'bindingInfo' should be placed in the onLoad handler.

Example (Report)

Bind a bar-type chart to the sample 'All Sales' query, with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

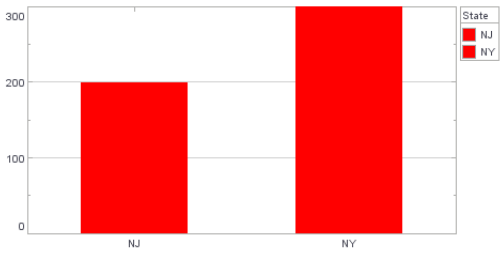
```
Chart1.bindingInfo.setColorField("Employee",Chart.STRING);
Chart1.bindingInfo.colorFrame = new CategoricalColorFrame;
Chart1.bindingInfo.colorFrame.setColor("Robert",java.awt.Co
lor.lightGray);
var robColor =
Chart1.bindingInfo.colorFrame.getColor("Robert")
Chart1.bindingInfo.colorFrame.setColor("Sue",robColor);
```



Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State","Quantity"],["NJ",200],["NY",300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity")
frame = new CategoricalColorFrame();
frame.setField("State");
frame.setColor('NJ',java.awt.Color(0xff0000));
var NJcolor = frame.getColor('NJ');
frame.setColor('NY',NJcolor);
elem.setColorFrame(frame);
graph.addElement(elem);
```



Inherits from

VisualFrame: setField, setLegendSpec, setScale, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

CategoricalColorFrame.INIT(VAL[,COLOR])

Initialize the ColorFrame with a set of categorical values and (optionally) colors.

Parameter

val	array of categorical initialization values, or name of field containing categorical values
color	array of colors corresponding to value array

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example 1 (Report or Viewsheet)

```
dataset = [["State","Quantity","Total"], ["NJ",200,10000],
["NY",300,8000]];
graph = new EGraph();
var elem = new PointElement("State", "Quantity");
var elem2 = new PointElement("State", "Total");
var cframe = new CategoricalColorFrame();
cframe.init("Quantity", "Total");
elem.setColorFrame(cframe);
elem2.setColorFrame(cframe);
var sscale = new CategoricalScale("State");
var qscale = new LinearScale("Quantity");
var qscale2 = new LinearScale("Total");
var coord = new RectCoord();
coord.setXScale(sscale);
coord.setYScale(qscale);
coord.setYScale2(qscale2);
graph.setCoordinate(coord);
graph.addElement(elem);
graph.addElement(elem2);
```



Example 2 (Report or Viewsheet)

```
dataset = [["State","Quantity","Total"], ["NJ",200,10000],
["NY",300,8000]];
graph = new EGraph();
var elem = new PointElement("State", "Quantity");
var elem2 = new PointElement("State", "Total");
var cframe = new CategoricalColorFrame();
cframe.init(["Quantity","Total"],[java.awt.Color(0xff00ff)
,java.awt.Color(0x00ffff)]);
```

```
elem.setColorFrame(cframe);
elem2.setColorFrame(cframe);
var sscale = new CategoricalScale("State");
var qscale = new LinearScale("Quantity");
var qscale2 = new LinearScale("Total");
var coord = new RectCoord();
coord.setXScale(sscale);
coord.setYScale(qscale);
coord.setYScale2(qscale2);
graph.setCoordinate(coord);
graph.addElement(elem);
graph.addElement(elem2);
```



Inherits from

VisualFrame: setField, setLegendSpec, setScale, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

CategoricalColorFrame.SETCOLOR(VAL,COLOR)

Assigns a color to the specified value.

Parameter

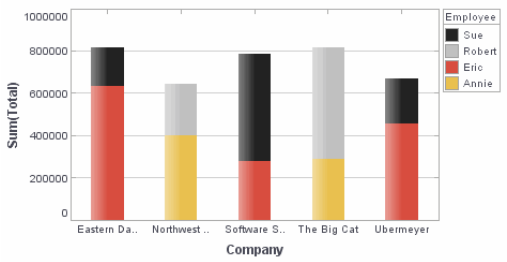
val	data value
color	java.awt.Color to represent data value

```
Note: Report script that modifies 'bindingInfo' should be placed in
the onLoad handler.
```

Example (Report)

Bind a bar-type chart to the sample ‘All Sales’ query, with ‘Company’ (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

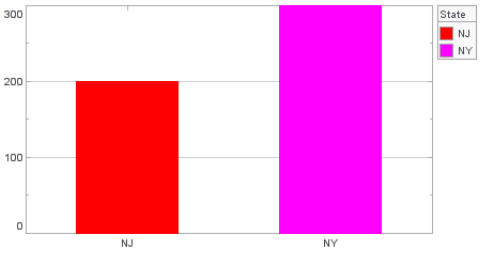
```
Chart1.bindingInfo.setColorField("Employee",Chart.STRING);
Chart1.bindingInfo.colorFrame = new CategoricalColorFrame;
Chart1.bindingInfo.colorFrame.setColor("Robert",java.awt.C
olor.lightGray);
Chart1.bindingInfo.colorFrame.setColor("Sue",java.awt.Colo
r(0x222222));
```



Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State","Quantity"], ["NJ",200], ["NY",300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity")
frame = new CategoricalColorFrame();
frame.setField("State");
frame.setColor('NJ',java.awt.Color(0xff0000));
frame.setColor('NY',java.awt.Color(0xff00ff));
elem.setColorFrame(frame)
graph.addElement(elem);
```



Inherits from

VisualFrame: setField, setLegendSpec, setScale, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.7.13 SizeFrame

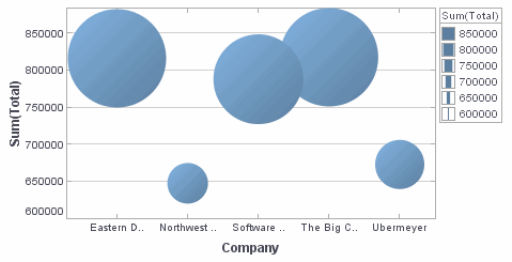
The SizeFrame object contains the size scale for visual chart objects. You can use a SizeFrame object to represent data dimensions with size (size coding), or to apply a fixed (static) size.

Note: Report script that modifies 'bindingInfo' should be placed in the onLoad handler.

Example (Report)

Bind a point-type chart to the sample ‘All Sales’ query, with ‘Company’ (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

```
Chart1.bindingInfo.setSizeField("Total",Chart.NUMBER);
Chart1.bindingInfo.sizeFrame = new LinearSizeFrame;
Chart1.bindingInfo.sizeFrame.smallest = 10;
Chart1.bindingInfo.sizeFrame.largest = 50;
Chart1.bindingInfo.sizeFrame.max = 100;
```

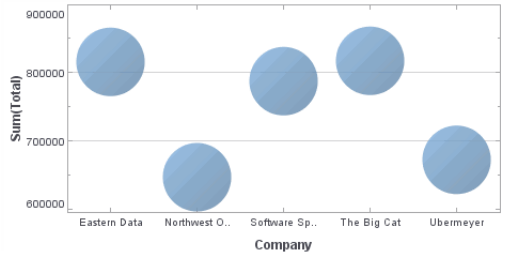


Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

Bind a point-type chart to the sample 'All Sales' query, with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis.

```
var frame = new StaticSizeFrame;
frame.setSize(30);
var elem = Chart1.graph.getElement(0); // point elements
elem.setSizeFrame(frame);
```



Full Name

`inetsoft.graph.aesthetic.SizeFrame`

SizeFrame: setLargest, setMax, setSmallest

Created by

StaticSizeFrame
LinearSizeFrame
CategoricalSizeFrame

Inherits from

VisualFrame: setField, setLegendSpec, setScale, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

SizeFrame.SETLARGEST(VALUE)

Specifies the highest value in the range. For point and line graphs, this is the size (in pixels) at which the largest value in the bound column is displayed. Smaller values are scaled according to the mapping defined by the object's scaling method (e.g., linear for LinearSizeFrame, etc.).

For schema and bar graphs, the value specified for the `largest` property is relative to the *SizeFrame.setMax(value)* value. For example, if `largest=50` and `max=100`, the largest value in the bound column is displayed at one half of the maximum allowable size. The value of `largest` should be less than the value of `max`.

Parameter

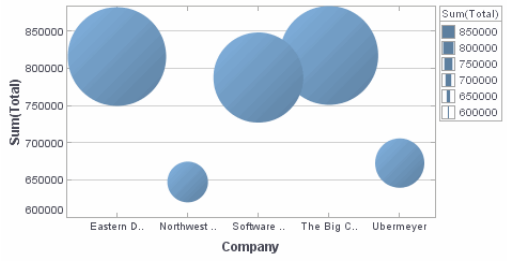
value	Number of pixels, or relative size

Note: Report script that modifies 'bindingInfo' should be placed in the onLoad handler.

Example (Report)

Bind a point-type chart to the sample 'All Sales' query, with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

```
Chart1.bindingInfo.setSizeField("Total",Chart.NUMBER);
Chart1.bindingInfo.sizeFrame = new LinearSizeFrame;
Chart1.bindingInfo.sizeFrame.smallest = 10;
Chart1.bindingInfo.sizeFrame.largest = 50;
Chart1.bindingInfo.sizeFrame.max = 100;
```

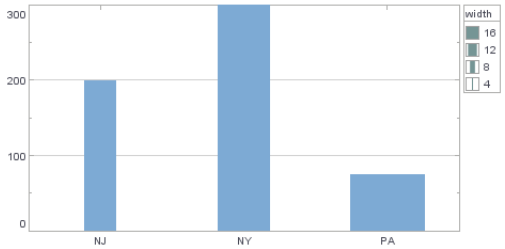


Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity", "width"], ["NJ", 200,
5],["NY", 300,10], ["PA",75,15]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity")
elem.sizeFrame = new LinearSizeFrame();
elem.sizeFrame.setField("width");
elem.sizeFrame.setSmallest(10);
elem.sizeFrame.setLargest(50);
```

```
elem.sizeFrame.setMax(100);
graph.addElement(elem);
```



Inherits from

VisualFrame: setField, setLegendSpec, setScale, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

SizeFrame.SETMAX(VALUE)

Specifies an arbitrary value representing the maximum allowable size for a graphical element. For schema and bar graphs, the values specified for the *SizeFrame.setLargest(value)* and *SizeFrame.setSmallest(value)* properties are relative to the `max` value. For example, if `smallest=50` and `max=100`, the smallest value in the bound column is displayed at one half of the maximum allowable size. For point and line graphs, the `max` property is ignored.

Parameter

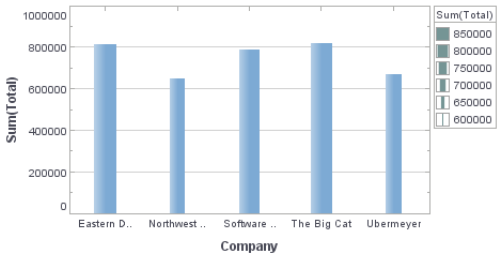
value	Number of pixels, or relative size

```
Note: Report script that modifies 'bindingInfo' should be placed in
the onLoad handler.
```

Example (Report)

Bind a point-type chart to the sample ‘All Sales’ query, with ‘Company’ (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

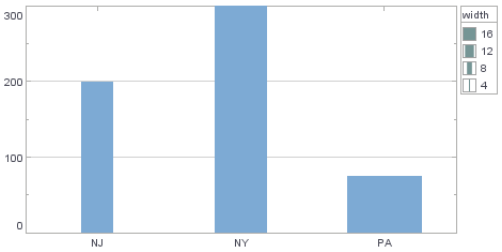
```
Chart1.bindingInfo.setSizeField("Total",Chart.NUMBER);
Chart1.bindingInfo.sizeFrame = new LinearSizeFrame;
Chart1.bindingInfo.sizeFrame.smallest = 10;
Chart1.bindingInfo.sizeFrame.largest = 50;
Chart1.bindingInfo.sizeFrame.max = 100;
```



Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity", "width"], ["NJ", 200,
5],["NY", 300,10], ["PA",75,15]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity")
elem.sizeFrame = new LinearSizeFrame();
elem.sizeFrame.setField("width");
elem.sizeFrame.setSmallest(10);
elem.sizeFrame.setLargest(50);
elem.sizeFrame.setMax(100);
graph.addElement(elem);
```



Inherits from

VisualFrame: setField, setLegendSpec, setScale, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

SizeFrame.SETSMALLEST(VALUE)

Specifies the lowest value in the range. For point and line graphs, this is the size (in pixels) at which the smallest value in the bound column is displayed. Larger values are scaled according to the mapping defined by the object's scaling method (e.g., linear for LinearSizeFrame, etc.).

For schema and bar graphs, the value specified for the `smallest` property is relative to the *SizeFrame.setMax(value)* value. For example, if `smallest=50` and `max=100`, the smallest value in the bound column is displayed at one half of the maximum allowable size.

Parameter

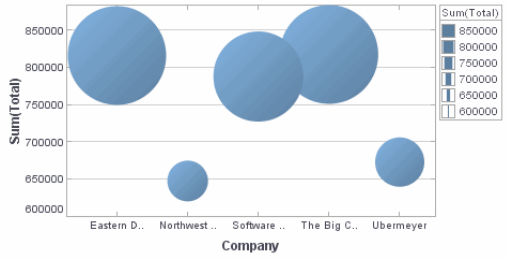
value Number of pixels, or relative size

Note: Report script that modifies 'bindingInfo' should be placed in the onLoad handler.

Example (Report)

Bind a point-type chart to the sample 'All Sales' query, with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

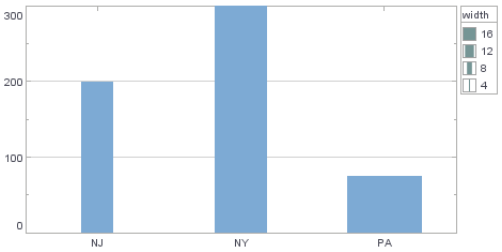
```
Chart1.bindingInfo.setSizeField("Total",Chart.NUMBER);
Chart1.bindingInfo.sizeFrame = new LinearSizeFrame;
Chart1.bindingInfo.sizeFrame.smallest = 10;
Chart1.bindingInfo.sizeFrame.largest = 50;
Chart1.bindingInfo.sizeFrame.max = 100;
```



Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity", "width"], ["NJ", 200,
5],["NY", 300,10], ["PA",75,15]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity")
elem.sizeFrame = new LinearSizeFrame();
elem.sizeFrame.setField("width");
elem.sizeFrame.setSmallest(10);
elem.sizeFrame.setLargest(50);
elem.sizeFrame.setMax(100);
graph.addElement(elem);
```



Inherits from

VisualFrame: setField, setLegendSpec, setScale, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.7.14 StaticSizeFrame

The StaticSizeFrame object contains a size frame defined by explicit size data in the *VisualFrame.setField(field)* column, or by the fixed size in *StaticSizeFrame.setSize(value)*. To create a StaticSizeFrame object, call the StaticSizeFrame constructor.

```
var frame = new StaticSizeFrame(10);
```

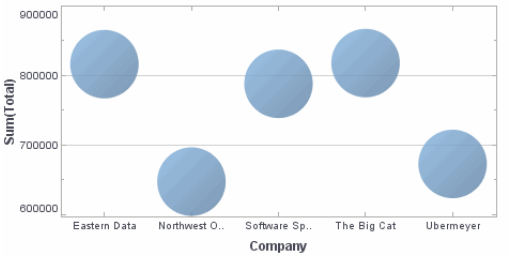
You can pass a numerical size (e.g., 10) to the constructor, or specify this later using the *StaticSizeFrame.setSize(value)* property.

Note: Report script that modifies 'bindingInfo' should be placed in the onLoad handler.

Example (Report)

Bind a point-type chart to the sample ‘All Sales’ query, with ‘Company’ (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

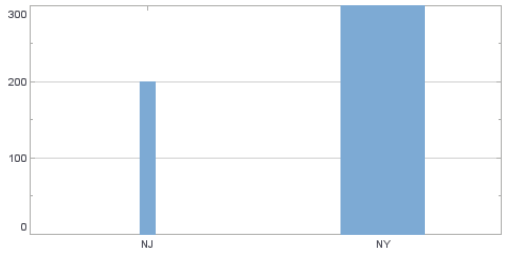
```
Chart1.bindingInfo.setSizeField("Company",Chart.STRING);
Chart1.bindingInfo.sizeFrame = new StaticSizeFrame;
Chart1.bindingInfo.sizeFrame.setSize(30);
```



Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State","Quantity","Size"],["NJ",200,1],
["NY",300,10]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var frame = new StaticSizeFrame();
frame.setField("Size");
elem.setSizeFrame(frame);
graph.addElement(elem);
```



Full Name

`inetsoft.graph.aesthetic.StaticSizeFrame`

StaticSizeFrame: setSize

Inherits from

SizeFrame: setLargest, setMax, setSmallest
VisualFrame: setField, setLegendSpec, setScale, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

StaticSizeFrame.SETSIZE(VALUE)

Specifies the static size for graphical elements. For point and line graphs, this is the size (in pixels) of the elements. For schema and bar graphs, the value specified for the `size` property is relative to the *SizeFrame.setMax(value)* value (default = 30). For example, if `size=50` and `max=100`, the graphical elements are displayed at one half of the maximum allowable size.

If the data in the column assigned to the inherited *VisualFrame.setField(field)* property are positive values, these data values are used instead of `StaticSizeFrame.size`.

Parameter

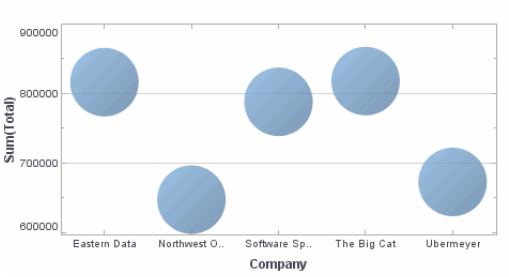
`value` `Number of pixels, or relative size`

```
Note: Report script that modifies 'bindingInfo' should be placed in
the onLoad handler.
```

Example (Report)

Bind a point-type chart to the sample ‘All Sales’ query, with ‘Company’ (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

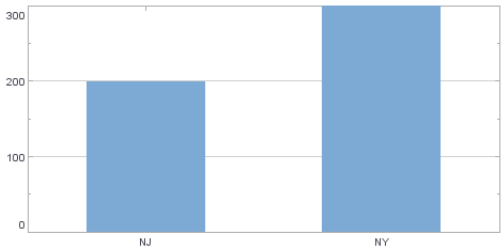
```
Chart1.bindingInfo.setSizeField("Company",Chart.STRING);
Chart1.bindingInfo.sizeFrame = new StaticSizeFrame;
Chart1.bindingInfo.sizeFrame.size = 30;
```



Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State","Quantity"], ["NJ",200], ["NY",300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity")
var frame = new StaticSizeFrame();
frame.setMax(100);
frame.setSize(50);
elem.setSizeFrame(frame);
graph.addElement(elem);
```



Inherits from

SizeFrame: setLargest, setMax, setSmallest
VisualFrame: setField, setLegendSpec, setScale, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.7.15 LinearSizeFrame

The LinearSizeFrame object contains a frame that linearly maps numerical data values to sizes. To create a LinearSizeFrame object, call the LinearSizeFrame constructor.

```
var frame = new LinearSizeFrame('Quantity');
```

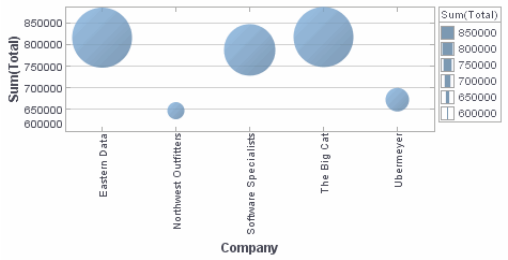
You can pass the name of a field (e.g., 'Quantity') to the constructor, or specify this later using the inherited *VisualFrame.setField(field)* property.

Note: Report script that modifies 'bindingInfo' should be placed in the onLoad handler.

Example (Report)

Bind a point-type chart to the sample 'All Sales' query, with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

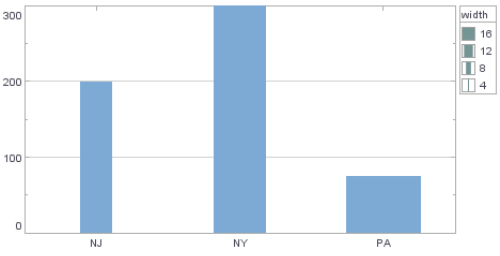
```
Chart1.bindingInfo.setSizeField("Total",Chart.NUMBER)
Chart1.bindingInfo.sizeFrame = new LinearSizeFrame;
```



Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity", "width"], ["NJ", 200, 5],
["NY", 300,10], ["PA",75,15]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var frame = new LinearSizeFrame();
frame.setField("width");
frame.setSmallest(10);
frame.setLargest(50);
frame.setMax(100);
elem.setSizeFrame(frame);
graph.addElement(elem);
```



Full Name

`inetsoft.graph.aesthetic.LinearSizeFrame`

Inherits from

SizeFrame: setLargest, setMax, setSmallest
VisualFrame: setField, setLegendSpec, setScale, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.7.16 CategoricalSizeFrame

The CategoricalSizeFrame object contains a frame that assigns a unique size to each distinct value. To create a CategoricalSizeFrame object, call the CategoricalSizeFrame constructor.

```
var frame = new CategoricalSizeFrame('State');
```

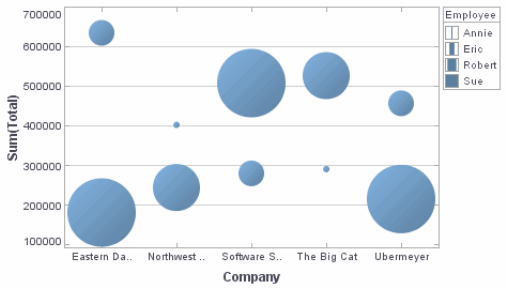
You can pass the name of a field (e.g., 'State') to the constructor, or specify this later using the inherited *VisualFrame.setField(field)* property.

Note: Report script that modifies 'bindingInfo' should be placed in the onLoad handler.

Example (Report)

Bind a point-type chart to the sample 'All Sales' query, with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

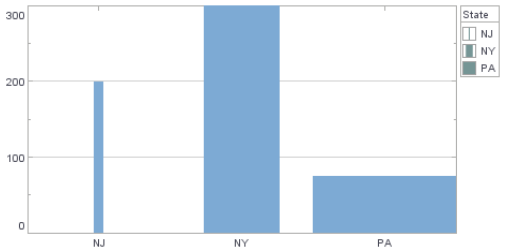
```
Chart1.bindingInfo.setSizeField("Employee",Chart.STRING)
Chart1.bindingInfo.sizeFrame = new CategoricalSizeFrame;
```



Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ", 200], ["NY", 300],
["PA", 75]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var frame = new CategoricalSizeFrame();
frame.setField("State");
elem.setSizeFrame(frame);
graph.addElement(elem);
```



Full Name

```
inetsoft.graph.aesthetic.CategoricalSizeFrame
```

CategoricalSizeFrame: setSize

Inherits from

SizeFrame: setLargest, setMax, setSmallest
VisualFrame: setField, setLegendSpec, setScale, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

CategoricalSizeFrame.SETSIZE(VAL,SIZE)

Assigns a size to the specified value.

Parameter

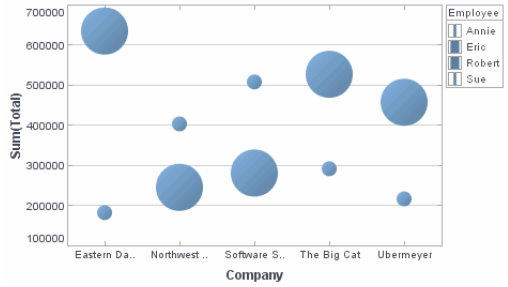
val	a data value
size	size at which to represent data value

```
Note: Report script that modifies 'bindingInfo' should be placed in
the onLoad handler.
```

Example (Report)

Bind a point-type chart to the sample 'All Sales' query, with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

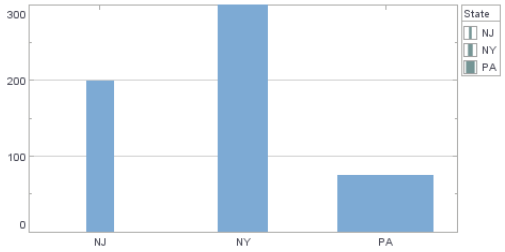
```
Chart1.bindingInfo.setSizeField("Employee",Chart.STRING)
Chart1.bindingInfo.sizeFrame = new CategoricalSizeFrame;
Chart1.bindingInfo.sizeFrame.setSize('Robert',20);
Chart1.bindingInfo.sizeFrame.setSize('Eric',20)
Chart1.bindingInfo.sizeFrame.setSize('Sue',5);
Chart1.bindingInfo.sizeFrame.setSize('Annie',5);
```



Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ", 200],["NY", 300],
["PA", 75]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity")
var frame = new CategoricalSizeFrame();
frame.setField("State");
frame.setSize('NJ',5);
frame.setSize('NY',10);
frame.setSize('PA',20);
elem.setSizeFrame(frame);
graph.addElement(elem);
```



Inherits from

SizeFrame: setLargest, setMax, setSmallest
VisualFrame: setField, setLegendSpec, setScale, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.7.17 TextureFrame

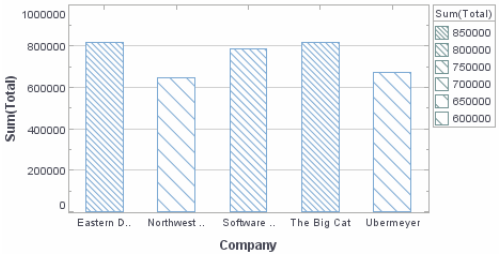
The TextureFrame object contains the texture for visual chart objects. You can use a TextureFrame object to represent data dimensions with texture (texture coding), or to apply a fixed (static) texture pattern.

Note: Report script that modifies 'bindingInfo' should be placed in the onLoad handler.

Example (Report)

Bind a point-type chart to the sample ‘All Sales’ query, with ‘Company’ (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

```
Chart1.bindingInfo.setShapeField("Total",Chart.NUMBER);
Chart1.bindingInfo.textureFrame = new LeftTiltTextureFrame;
```

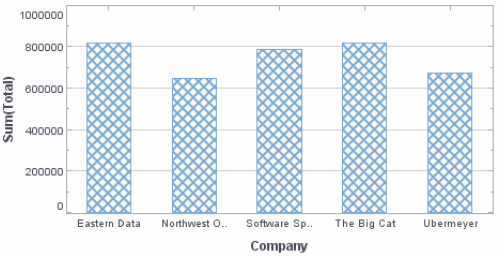


Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report)

Bind a point-type chart to the sample 'All Sales' query, with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis.

```
var frame = new StaticTextureFrame();
frame.setTexture(GTexture.PATTERN_5);
var elem = graph.getElement(0); // bar elements
elem.setTextureFrame(frame);
```



Full Name

`inetsoft.graph.aesthetic.TextureFrame`

Created by

LeftTiltTextureFrame
OrientationTextureFrame
RightTiltTextureFrame
GridTextureFrame
CategoricalTextureFrame
StaticTextureFrame

Inherits from

VisualFrame: setField, setLegendSpec, setScale, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.7.18 StaticTextureFrame

The StaticTextureFrame object contains a texture frame defined by explicit texture data in the *VisualFrame.setField(field)* column, or by the fixed texture in *StaticTextureFrame.setTexture(value)*. To create a StaticTextureFrame object, call the StaticTextureFrame constructor.

```
var frame = new StaticTextureFrame();
```

You can pass a *GTexture* object directly to the constructor, e.g.,

```
var frame = new StaticTextureFrame(GTexture.PATTERN_5);
```

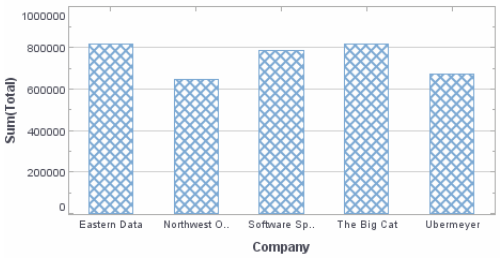
or specify it later using the *StaticTextureFrame.setTexture(value)* property.

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report)

Bind a point-type chart to the sample 'All Sales' query, with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis.

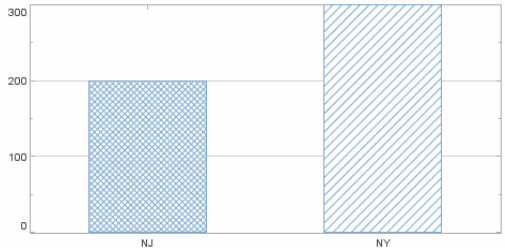
```
var frame = new StaticTextureFrame();
frame.setTexture(GTexture.PATTERN_5);
var elem = graph.getElement(0); // bar elements
elem.setTextureFrame(frame);
```



Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State","Quantity","Texture"],["NJ",200,1],
["NY",300,10]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var frame = new StaticTextureFrame();
frame.setField("Texture");
elem.setTextureFrame(frame);
graph.addElement(elem);
```



Full Name

`inetsoft.graph.aesthetic.StaticTextureFrame`

StaticTextureFrame: setTexture

Inherits from

VisualFrame: setField, setLegendSpec, setScale, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

StaticTextureFrame.SETTEXTURE(VALUE)

Specifies the static texture for graphical elements. If the data in the column assigned to the inherited *VisualFrame.setField(field)* property are `GTexture` numbers, these data values are used instead of `StaticTextureFrame.texture`.

Parameter

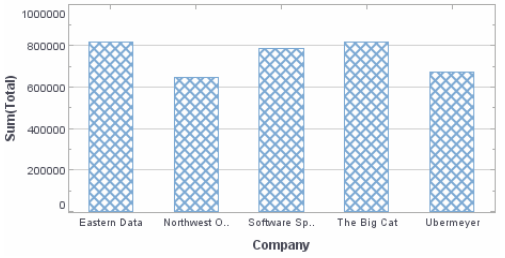
value a `GTexture` object

```
Note: Report script that modifies 'bindingInfo' should be placed in
the onLoad handler.
```

Example (Report)

Bind a point-type chart to the sample ‘All Sales’ query, with ‘Company’ (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

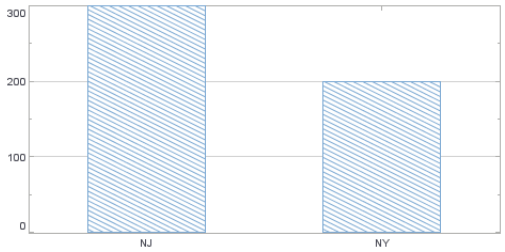
```
Chart1.bindingInfo.setShapeField("Company",Chart.STRING);
Chart1.bindingInfo.textureFrame = new StaticTextureFrame();
Chart1.bindingInfo.textureFrame.texture =
GTexture.PATTERN_5;
```



Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State","Quantity"], ["NJ",300], ["NY",200]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity")
frame = new StaticTextureFrame();
frame.setTexture(GTexture.PATTERN_18);
elem.setTextureFrame(frame);
graph.addElement(elem);
```



Inherits from

VisualFrame: setField, setLegendSpec, setScale, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.7.19 LeftTiltTextureFrame

The LeftTiltTextureFrame object contains a texture for visual chart objects rendered by negatively-sloping lines with variable spacing. To create a LeftTiltTextureFrame object, call the LeftTiltTextureFrame constructor.

```
var frame = new LeftTiltTextureFrame('Total');
```

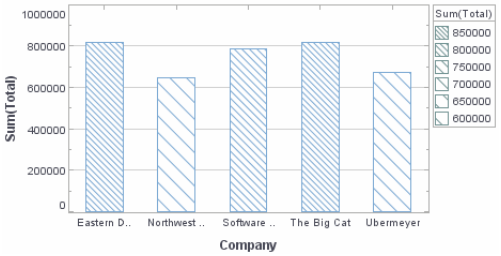
You can pass the name of a field (e.g., 'Total') to the constructor, or specify this later using the inherited *VisualFrame.setField(field)* property.

Note: Report script that modifies 'bindingInfo' should be placed in the onLoad handler.

Example (Report)

Bind a point-type chart to the sample 'All Sales' query, with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

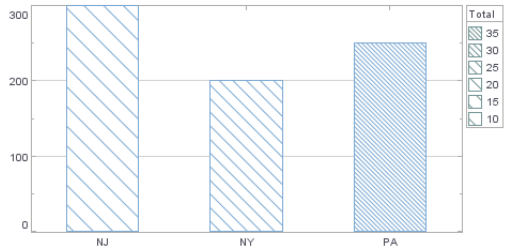
```
Chart1.bindingInfo.setShapeField("Total",Chart.NUMBER);
Chart1.bindingInfo.textureFrame = new LeftTiltTextureFrame;
```



Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity","Total"],
["NJ",300,10],["NY",200,20],["PA",250,35]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity")
var frame = new LeftTiltTextureFrame("Total");
elem.setTextureFrame(frame);
graph.addElement(elem);
```



Full Name

`inetsoft.graph.aesthetic.LeftTiltTextureFrame`

Inherits from

VisualFrame: setField, setLegendSpec, setScale, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.7.20 OrientationTextureFrame

The OrientationTextureFrame object contains a texture for visual chart objects rendered by uniformly-spaced lines with variable slope. To create a OrientationTextureFrame object, call the OrientationTextureFrame constructor.

```
var frame = new OrientationTextureFrame('Total');
```

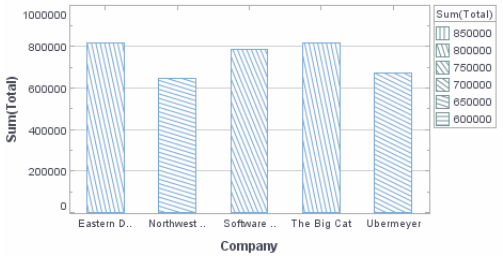
You can pass the name of a field (e.g., 'Total') to the constructor, or specify this later using the inherited *VisualFrame.setField(field)* property.

Note: Report script that modifies 'bindingInfo' should be placed in the onLoad handler.

Example (Report)

Bind a point-type chart to the sample 'All Sales' query, with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

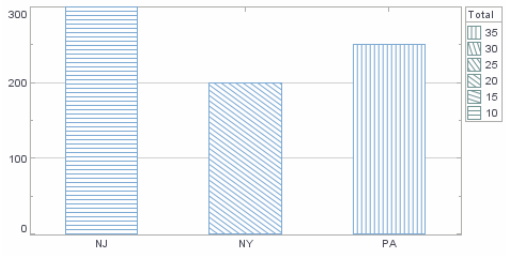
```
Chart1.bindingInfo.setShapeField("Total",Chart.NUMBER);
Chart1.bindingInfo.textureFrame = new
OrientationTextureFrame;
```



Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity","Total"],
["NJ",300,10],["NY",200,20],["PA",250,35]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity")
var frame = new OrientationTextureFrame("Total");
elem.setTextureFrame(frame);
graph.addElement(elem);
```



Full Name

inetsoft.graph.aesthetic.OrientationTextureFrame

Inherits from

VisualFrame: setField, setLegendSpec, setScale, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.7.21 RightTiltTextureFrame

The RightTiltTextureFrame object contains a texture for visual chart objects rendered by positively-sloping lines with variable spacing. To create a RightTiltTextureFrame object, call the RightTiltTextureFrame constructor.

```
var frame = new RightTiltTextureFrame('Total');
```

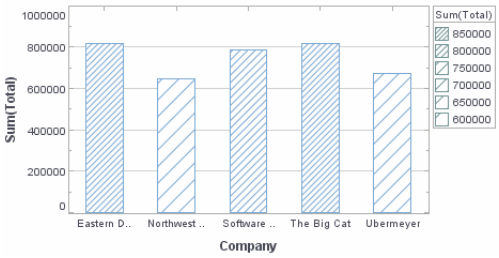
You can pass the name of a field (e.g., 'Total') to the constructor, or specify this later using the inherited *VisualFrame.setField(field)* property.

Note: Report script that modifies 'bindingInfo' should be placed in the onLoad handler.

Example (Report)

Bind a point-type chart to the sample 'All Sales' query, with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

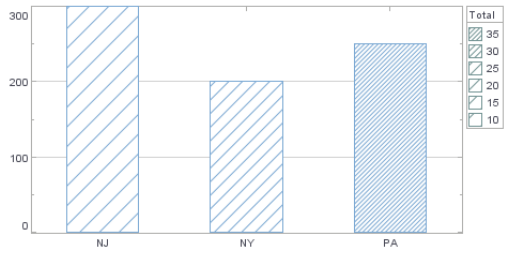
```
Chart1.bindingInfo.setShapeField("Total",Chart.NUMBER);
Chart1.bindingInfo.textureFrame = new
RightTiltTextureFrame;
```



Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity","Total"],
["NJ",300,10],["NY",200,20],["PA",250,35]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity")
var frame = new RightTiltTextureFrame("Total");
elem.setTextureFrame(frame);
graph.addElement(elem);
```



Full Name

```
inetsoft.graph.aesthetic.RightTiltTextureFrame
```

Inherits from

VisualFrame: setField, setLegendSpec, setScale, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.7.22 GridTextureFrame

The GridTextureFrame object contains a texture for visual chart objects rendered by variably-spaced orthogonal lines. To create a GridTextureFrame object, call the GridTextureFrame constructor.

```
var frame = new GridTextureFrame('Total');
```

You can pass the name of a field (e.g., 'Total') to the constructor, or specify this later using the inherited *VisualFrame.setField(field)* property.

Full Name

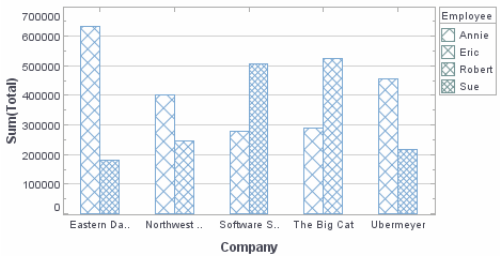
```
inetsoft.graph.aesthetic.GridTextureFrame
```

```
Note: Report script that modifies 'bindingInfo' should be placed in
the onLoad handler.
```

Example (Report)

Bind a point-type chart to the sample 'All Sales' query, with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

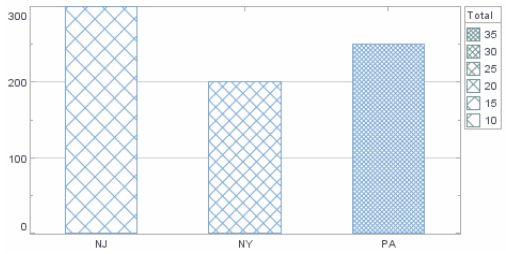
```
Chart1.bindingInfo.setShapeField("Employee",Chart.STRING);
Chart1.bindingInfo.textureFrame = new GridTextureFrame();
```



Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity","Total"],
["NJ",300,10],["NY",200,20],["PA",250,35]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity")
var frame = new GridTextureFrame("Total");
elem.setTextureFrame(frame);
graph.addElement(elem);
```



Full Name

```
inetsoft.graph.aesthetic.GridTextureFrame
```

Inherits from

VisualFrame: setField, setLegendSpec, setScale, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.7.23 CategoricalTextureFrame

The CategoricalTextureFrame object contains a texture for visual chart objects, rendering a unique texture for each discrete value. To create a CategoricalTextureFrame object, call the CategoricalTextureFrame constructor.

```
var frame = new GridTextureFrame('State');
```

You can pass the name of a field (e.g., 'State') to the constructor, or specify this later using the inherited *VisualFrame.setField(field)* property.

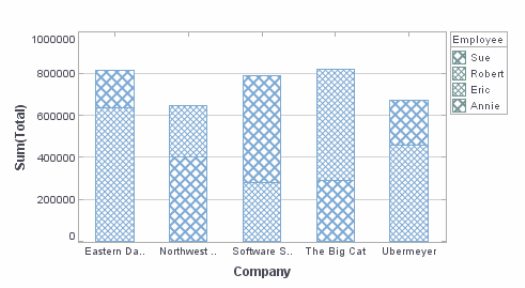
Note: Report script that modifies 'bindingInfo' should be placed in the onLoad handler.

Example (Report)

Bind a point-type chart to the sample 'All Sales' query, with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

```
Chart1.bindingInfo.setShapeField("Employee",Chart.STRING);
```

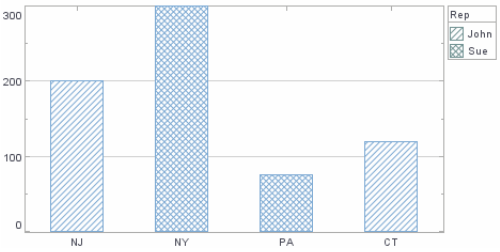
```
Chart1.bindingInfo.textureFrame = new
CategoricalTextureFrame;
Chart1.bindingInfo.textureFrame.setTexture('Robert',GTextur
e.PATTERN_1);
Chart1.bindingInfo.textureFrame.setTexture('Eric',GTexture.
PATTERN_1);
Chart1.bindingInfo.textureFrame.setTexture('Sue',GTexture.P
ATTERN_5);
Chart1.bindingInfo.textureFrame.setTexture('Annie',GTexture
.PATTERN_5);
```



Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State","Quantity","Rep"], ["NJ",200,"John"],
["NY",300,"Sue"], ["PA",75,"Sue"], ["CT", 120,"John"]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity")
var frame = new CategoricalTextureFrame();
frame.setField("Rep");
elem.setTextureFrame(frame);
graph.addElement(elem);
```



Full Name

inetsoft.graph.aesthetic.CategoricalTextureFrame

Inherits from

VisualFrame: setField, setLegendSpec, setScale, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

CategoricalTextureFrame: setTexture

CategoricalTextureFrame.SETTEXTURE(VAL,TEXTURE)

Assigns a texture to the specified value.

Parameter

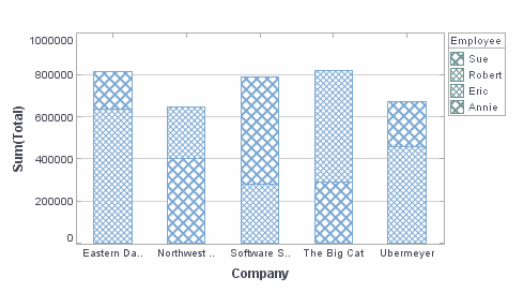
val	a data value
shape	a *GTexture* object

Note: Report script that modifies 'bindingInfo' should be placed in the onLoad handler.

Example (Report)

Bind a point-type chart to the sample 'All Sales' query, with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

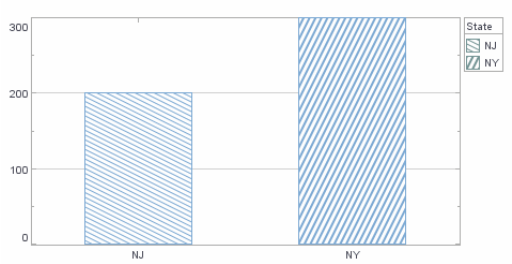
```
Chart1.bindingInfo.setShapeField("Employee",Chart.STRING);
Chart1.bindingInfo.textureFrame = new
CategoricalTextureFrame;
Chart1.bindingInfo.textureFrame.setTexture('Robert',GTextu
re.PATTERN_1);
Chart1.bindingInfo.textureFrame.setTexture('Eric',GTexture
.PATTERN_1);
Chart1.bindingInfo.textureFrame.setTexture('Sue',GTexture.
PATTERN_5);
Chart1.bindingInfo.textureFrame.setTexture('Annie',GTextur
e.PATTERN_5);
```



Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State","Quantity"], ["NJ",200], ["NY",300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
textureframe = new CategoricalTextureFrame("State");
textureframe.setTexture('NJ',GTexture.PATTERN_18);
textureframe.setTexture('NY',GTexture.PATTERN_14);
elem.setTextureFrame(textureframe);
graph.addElement(elem);
```



Inherits from
VisualFrame: setField, setLegendSpec, setScale, setScaleOption

See Also
DataSet, to use a Data Block or query as the *dataset* source.

CR.7.24 ShapeFrame

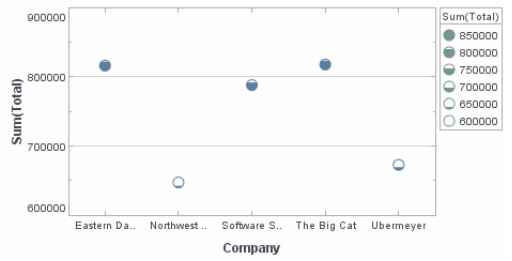
A ShapeFrame object contains the shape style for visual chart objects. You can use a ShapeFrame object to represent data dimensions with shape (shape coding), or to apply a fixed (static) shape style.

```
Note: Report script that modifies 'bindingInfo' should be placed in
the onLoad handler.
```

Example (Report)

Bind a point-type chart to the sample ‘All Sales’ query, with ‘Company’ (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

```
Chart1.bindingInfo.setShapeField("Total",Chart.NUMBER);
Chart1.bindingInfo.shapeFrame = new FillShapeFrame;
```



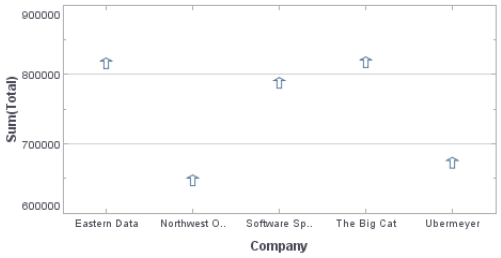

```
Note: Script that modifies 'graph' should be placed at the element
level. See Adding Element-Level Script in Report Scripting and
Adding Component Script in Dashboard Scripting for more
information.
```

Example (Report or Viewsheet)

Bind a point-type chart to the sample ‘All Sales’ query, with ‘Company’ (top 5) on the X-axis, and Sum(Total) on the Y-axis.

```
var frame = new StaticShapeFrame();
```

```
frame.setShape(GShape.ARROWBAR);
var elem = graph.getElement(0); // point elements
elem.setShapeFrame(frame);
```



Full Name

`inetsoft.graph.aesthetic.ShapeFrame`

Created by

OvalShapeFrame
FillShapeFrame
OrientationShapeFrame
PolygonShapeFrame
TriangleShapeFrame
CategoricalShapeFrame
StaticShapeFrame

Inherits from

VisualFrame: setField, setLegendSpec, setScale, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.7.25 OvalShapeFrame

The OvalShapeFrame object contains the shape styles for oval elements of varying aspect ratio. To create a OvalShapeFrame object, call the OvalShapeFrame constructor.

```
var frame = new OvalShapeFrame('Total');
```

You can pass the name of a field (e.g., 'Total') to the constructor, or specify this later using the inherited *VisualFrame.setField(field)* property.

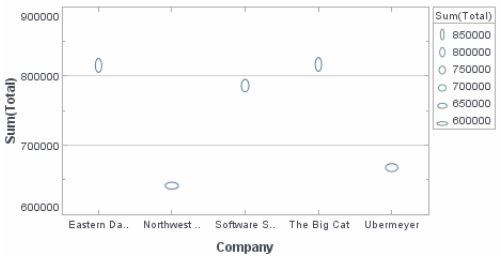
```
Note: Report script that modifies 'bindingInfo' should be placed in
the onLoad handler.
```

Example (Report)

Bind a point-type chart to the sample 'All Sales' query, with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

```
Chart1.bindingInfo.setShapeField("Total",Chart.NUMBER);
```

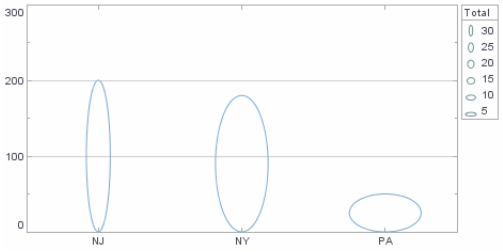
```
Chart1.bindingInfo.shapeFrame = new OvalShapeFrame();
```



Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State","Quantity","Total"], ["NJ",200,30],
["NY",300,15],["PA",150,5]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var frame = new OvalShapeFrame("Total");
elem.setShapeFrame(frame);
graph.addElement(elem);
```



Full Name

```
inetsoft.graph.aesthetic.OvalShapeFrame
```

OvalShapeFrame: setFill

Inherits from

VisualFrame: setField, setLegendSpec, setScale, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

OvalShapeFrame.SETFILL(VALUE)

Specifies the fill for the oval elements.

Parameter

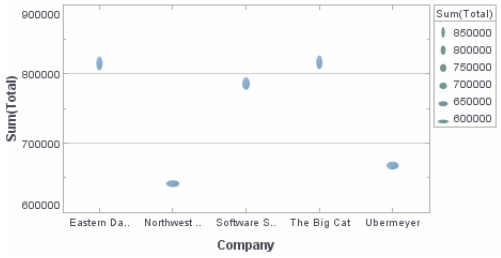
boolean	true: fill oval false: do not fill oval

Note: Report script that modifies 'bindingInfo' should be placed in the onLoad handler.

Example (Report)

Bind a point-type chart to the sample 'All Sales' query, with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

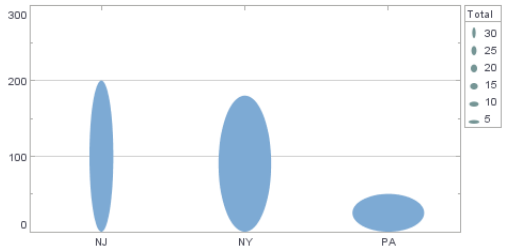
```
Chart1.bindingInfo.setShapeField("Total",Chart.NUMBER);
Chart1.bindingInfo.shapeFrame = new OvalShapeFrame;
Chart1.bindingInfo.shapeFrame.fill = true;
```



Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State","Quantity","Total"], ["NJ",200,30],
["NY",300,15],["PA",150,5]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var frame = new OvalShapeFrame("Total");
frame.setFill(true);
elem.setShapeFrame(frame);
graph.addElement(elem);
```



Inherits from

VisualFrame: setField, setLegendSpec, setScale, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.7.26 FillShapeFrame

The FillShapeFrame object contains the shape styles for oval elements with variable degree of fill. To create a FillShapeFrame object, call the FillShapeFrame constructor.

```
var frame = new FillShapeFrame('Total');
```

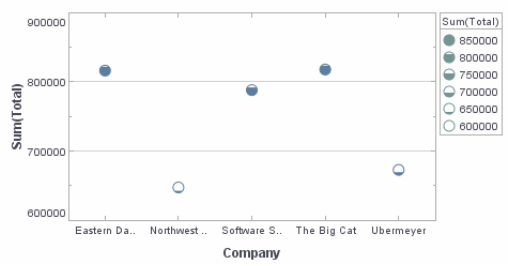
You can pass the name of a field (e.g., 'Total') to the constructor, or specify this later using the inherited *VisualFrame.setField(field)* property.

Note: Report script that modifies 'bindingInfo' should be placed in the onLoad handler.

Example (Report)

Bind a point-type chart to the sample 'All Sales' query, with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

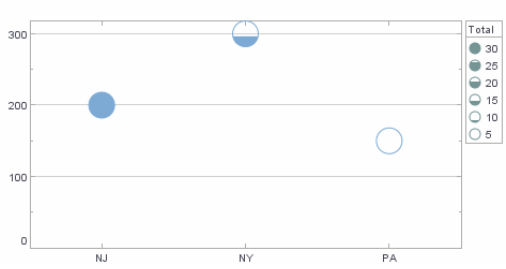
```
Chart1.bindingInfo.setShapeField("Total",Chart.NUMBER);
Chart1.bindingInfo.shapeFrame = new FillShapeFrame;
```



Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State","Quantity","Total"], ["NJ",200,30],
["NY",300,15], ["PA",150,5]];
graph = new EGraph();
var elem = new PointElement("State", "Quantity");
var shapeframe = new FillShapeFrame("Total");
var sizeframe = new StaticSizeFrame();
sizeframe.setSize(10);
elem.setSizeFrame(sizeframe);
elem.setShapeFrame(shapeframe);
graph.addElement(elem);
```



Full Name

`inetsoft.graph.aesthetic.FillShapeFrame`

Inherits from

VisualFrame: setField, setLegendSpec, setScale, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.7.27 OrientationShapeFrame

The OrientationShapeFrame object contains the shape styles for line elements with variable orientation. To create a OrientationShapeFrame object, call the OrientationShapeFrame constructor.

```
var frame = new OrientationShapeFrame('Total');
```

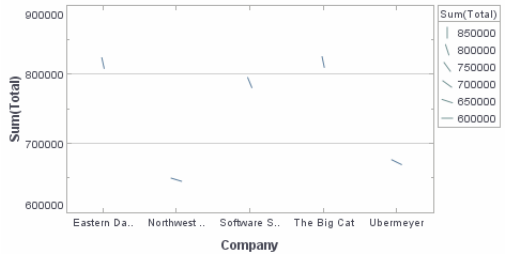
You can pass the name of a field (e.g., 'Total') to the constructor, or specify this later using the inherited *VisualFrame.setField(field)* property.

```
Note: Report script that modifies 'bindingInfo' should be placed in
the onLoad handler.
```

Example (Report)

Bind a point-type chart to the sample 'All Sales' query, with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

```
Chart1.bindingInfo.setShapeField("Total",Chart.NUMBER);
Chart1.bindingInfo.shapeFrame = new OrientationShapeFrame;
```




```
Note: Script that modifies 'graph' should be placed at the element
level. See Adding Element-Level Script in Report Scripting and
```

Adding Component Script in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State","Quantity","Total"], ["NJ",200,30],
["NY",300,15], ["PA",150,5]];
graph = new EGraph();
var elem = new PointElement("State", "Quantity");
var shapeframe = new OrientationShapeFrame("Total");
var sizeframe = new StaticSizeFrame();
sizeframe.setSize(10);
elem.setSizeFrame(sizeframe);
elem.setShapeFrame(shapeframe);
graph.addElement(elem);
```



Full Name

```
inetsoft.graph.aesthetic.OrientationShapeFrame
```

Inherits from

VisualFrame: setField, setLegendSpec, setScale, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.7.28 PolygonShapeFrame

The PolygonShapeFrame object contains the shape styles for elements with varying number of sides. To create a PolygonShapeFrame object, call the PolygonShapeFrame constructor.

```
var frame = new PolygonShapeFrame('Total');
```

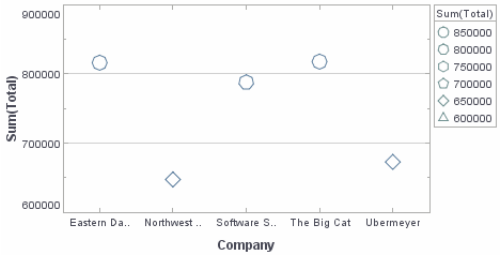
You can pass the name of a field (e.g., 'Total') to the constructor, or specify this later using the inherited *VisualFrame.setField(field)* property.

```
Note: Report script that modifies 'bindingInfo' should be placed in
the onLoad handler.
```

Example (Report)

Bind a point-type chart to the sample 'All Sales' query, with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

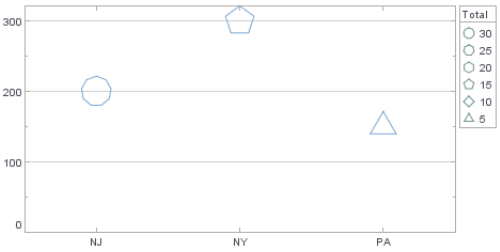
```
Chart1.bindingInfo.setShapeField("Total",Chart.NUMBER);
Chart1.bindingInfo.shapeFrame = new PolygonShapeFrame;
```



Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State","Quantity","Total"], ["NJ",200,30],
["NY",300,15], ["PA",150,5]];
graph = new EGraph();
var elem = new PointElement("State", "Quantity");
var sizeframe = new StaticSizeFrame();
sizeframe.setSize(10);
var frame = new PolygonShapeFrame("Total");
elem.setShapeFrame(frame);
elem.setSizeFrame(sizeframe);
graph.addElement(elem);
```



Full Name

`inetsoft.graph.aesthetic.PolygonShapeFrame`

PolygonShapeFrame: setFill

Inherits from

VisualFrame: setField, setLegendSpec, setScale, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

PolygonShapeFrame.SETFILL(BOOLEAN)

Specifies whether the polygonal elements should be filled.

Parameter

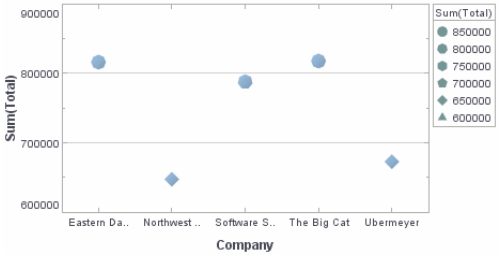
boolean	`true: fill oval` `false: do not fill oval`

Note: Report script that modifies 'bindingInfo' should be placed in the onLoad handler.

Example (Report)

Bind a point-type chart to the sample 'All Sales' query, with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

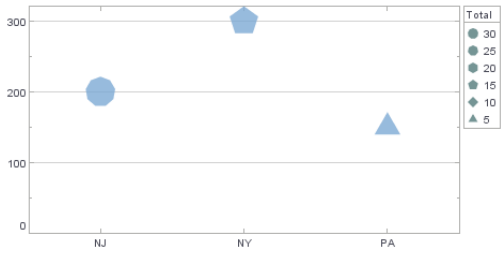
```
Chart1.bindingInfo.setShapeField("Total",Chart.NUMBER);
Chart1.bindingInfo.shapeFrame = new PolygonShapeFrame;
Chart1.bindingInfo.shapeFrame.fill = true;
```



Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State","Quantity","Total"], ["NJ",200,30],
["NY",300,15],["PA",150,5]];
graph = new EGraph();
var elem = new PointElement("State", "Quantity");
var sizeframe = new StaticSizeFrame();
sizeframe.setSize(10);
var frame = new PolygonShapeFrame("Total");
frame.setFill(true);
elem.setShapeFrame(frame);
elem.setSizeFrame(sizeframe);
graph.addElement(elem);
```



Inherits from

VisualFrame: setField, setLegendSpec, setScale, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.7.29 TriangleShapeFrame

The TriangleShapeFrame object contains the shape styles for isosceles trapezoid elements with varying width ratios. To create a TriangleShapeFrame object, call the TriangleShapeFrame constructor.

```
var frame = new TriangleShapeFrame('Total');
```

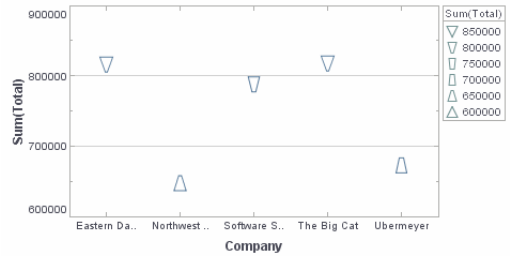
You can pass the name of a field (e.g., 'Total') to the constructor, or specify this later using the inherited *VisualFrame.setField(field)* property.

Note: Report script that modifies 'bindingInfo' should be placed in the onLoad handler.

Example (Report)

Bind a point-type chart to the sample 'All Sales' query, with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

```
Chart1.bindingInfo.setShapeField("Total",Chart.NUMBER);
Chart1.bindingInfo.shapeFrame = new TriangleShapeFrame;
```



Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State","Quantity","Total"], ["NJ",200,30],
["NY",300,15], ["PA",150,5]];
graph = new EGraph();
var elem = new PointElement("State", "Quantity");
var sizeframe = new StaticSizeFrame();
sizeframe.setSize(10);
var frame = new TriangleShapeFrame("Total");
elem.setShapeFrame(frame);
elem.setSizeFrame(sizeframe);
graph.addElement(elem);
```



Full Name

`inetsoft.graph.aesthetic.TriangleShapeFrame`

TriangleShapeFrame: setFill

Inherits from

VisualFrame: setField, setLegendSpec, setScale, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

TriangleShapeFrame.SETFILL(BOOLEAN)

Specifies whether the triangular elements should be filled.

Parameter

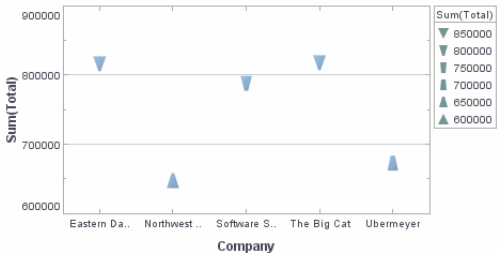
boolean `true: fill triangle`
`false: do not fill triangle`

```
Note: Report script that modifies 'bindingInfo' should be placed in
the onLoad handler.
```

Example (Report)

Bind a point-type chart to the sample ‘All Sales’ query, with ‘Company’ (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

```
Chart1.bindingInfo.setShapeField("Total",Chart.NUMBER);
Chart1.bindingInfo.shapeFrame = new TriangleShapeFrame;
Chart1.bindingInfo.shapeFrame.fill = true;
```

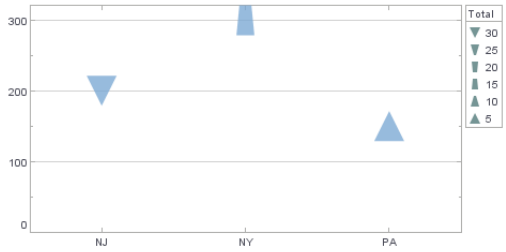



```
Note: Script that modifies 'graph' should be placed at the element
level. See Adding Element-Level Script in Report Scripting and
```

Adding Component Script in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State","Quantity","Total"], ["NJ",200,30],
["NY",300,15], ["PA",150,5]];
graph = new EGraph();
var elem = new PointElement("State", "Quantity");
var sizeframe = new StaticSizeFrame();
sizeframe.setSize(10);
var frame = new TriangleShapeFrame("Total");
frame.setFill(true);
elem.setShapeFrame(frame);
elem.setSizeFrame(sizeframe);
graph.addElement(elem);
```



Inherits from

VisualFrame: setField, setLegendSpec, setScale, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.7.30 CategoricalShapeFrame

The CategoricalShapeFrame object contains a shape style for visual chart objects, rendering a unique shape for each discrete value. To create a CategoricalShapeFrame object, call the CategoricalShapeFrame constructor.

```
var frame = new CategoricalShapeFrame('State');
```

You can pass the name of a field (e.g., 'State') to the constructor, or specify this later using the inherited *VisualFrame.setField(field)* property.

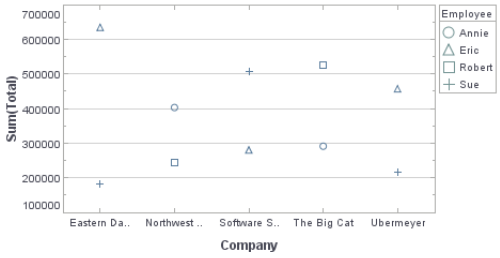
```
Note: Report script that modifies 'bindingInfo' should be placed in
the onLoad handler.
```

Example (Report)

Bind a point-type chart to the sample 'All Sales' query, with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

Example

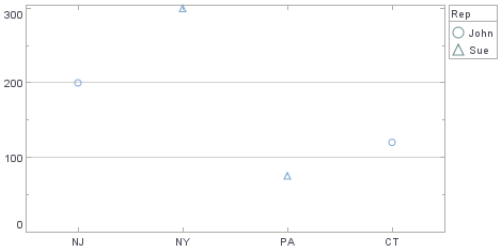
```
Chart1.bindingInfo.setShapeField("Employee",Chart.STRING);
Chart1.bindingInfo.shapeFrame = new CategoricalShapeFrame;
```



Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State","Quantity","Rep"], ["NJ",200,"John"],
["NY",300,"Sue"], ["PA",75,"Sue"], ["CT", 120,"John"]];
graph = new EGraph();
var elem = new PointElement("State", "Quantity")
var frame = new CategoricalShapeFrame("Rep");
elem.setShapeFrame(frame);
graph.addElement(elem);
```



Full Name

`inetsoft.graph.aesthetic.CategoricalShapeFrame`

Inherits from

VisualFrame: setField, setLegendSpec, setScale, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

CategoricalShapeFrame: setShape

CategoricalShapeFrame.SETSHAPE(VAL,SHAPE)

Assigns a shape to the specified value.

Parameter

val	data value
shape	*GShape* or *GShape.ImageShape* object

Note: Report script that modifies 'bindingInfo' should be placed in the onLoad handler.

Example (Report)

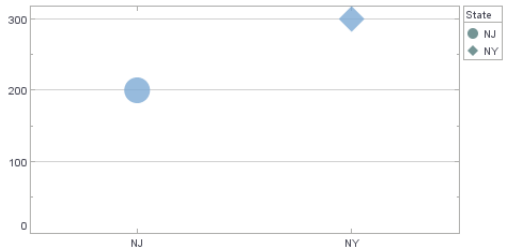
Bind a point-type chart to the sample 'All Sales' query, with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

```
Chart1.bindingInfo.setShapeField("Employee",Chart.STRING);
Chart1.bindingInfo.shapeFrame = new CategoricalShapeFrame;
Chart1.bindingInfo.shapeFrame.setShape('Robert',
GShape.FILLED_CIRCLE);
Chart1.bindingInfo.shapeFrame.setShape('Sue',
GShape.FILLED_DIAMOND);
```

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State","Quantity"], ["NJ",200], ["NY",300]];
graph = new EGraph();
var elem = new PointElement("State", "Quantity");
sizeframe = new StaticSizeFrame("State");
sizeframe.setSize(10);
shapeframe = new CategoricalShapeFrame("State");
shapeframe.setShape('NJ',GShape.FILLED_CIRCLE);
shapeframe.setShape('NY',GShape.FILLED_DIAMOND);
elem.setShapeFrame(shapeframe);
elem.setSizeFrame(sizeframe);
graph.addElement(elem);
```



Inherits from

VisualFrame: setField, setLegendSpec, setScale, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.7.31 StaticShapeFrame

The StaticShapeFrame object contains a shape frame defined by explicit shape data in the *VisualFrame.setField(field)* column, or by the fixed shape in *StaticShapeFrame.setShape(shape)*. To create a StaticShapeFrame object, call the StaticShapeFrame constructor.

```
var frame = new StaticShapeFrame('GShape.CIRCLE');
```

You can pass a *GShape* or *SVGShape* object (e.g., '`GShape.CIRCLE`', '`SVGShape.FACE_BLANK`') or a field name (e.g., 'Shape') to the constructor,

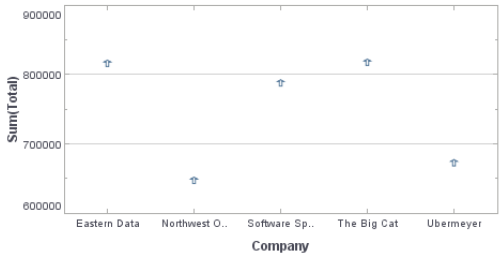
or specify this later using the *StaticShapeFrame.setShape(shape)* or inherited *VisualFrame.setField(field)* property.

Note: Report script that modifies 'bindingInfo' should be placed in the onLoad handler.

Example 1 (Report)

Bind a point-type chart to the sample 'All Sales' query, with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

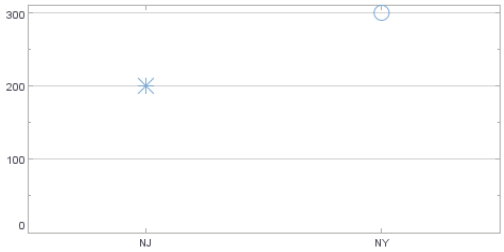
```
Chart1.bindingInfo.setShapeField("Company",Chart.STRING);
Chart1.bindingInfo.shapeFrame = new StaticShapeFrame;
Chart1.bindingInfo.shapeFrame.setShape(GShape.ARROWBAR);
```



Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example 2 (Report or Viewsheet)

```
dataset = [["State","Quantity","Shape"], ["NJ",200,"STAR"],
["NY",300,"CIRCLE"]];
graph = new EGraph();
var elem = new PointElement("State", "Quantity");
var sizeframe = new StaticSizeFrame(5);
var shapeframe = new StaticShapeFrame();
shapeframe.setField("Shape");
elem.setShapeFrame(shapeframe);
elem.setSizeFrame(sizeframe);
graph.addElement(elem);
```



Example 3 (Report or Viewsheet)

```
dataset = [["State","Quantity"], ["NJ",200], ["NY",300]];
graph = new EGraph();
var elem = new PointElement("State","Quantity");
```

```
var shapeFrame = new StaticShapeFrame(SVGShape.FACE_HAPPY);
var sizeFrame = new StaticSizeFrame(10);
elem.setSizeFrame(sizeFrame);
elem.setShapeFrame(shapeFrame);
graph.addElement(elem);
```



Full Name

`inetsoft.graph.aesthetic.StaticShapeFrame`

StaticShapeFrame: setShape

Inherits from

VisualFrame: setField, setLegendSpec, setScale, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

StaticShapeFrame.SETSHAPE(SHAPE)

Specifies the static shape for graphical elements. If the data in the column assigned to the inherited *VisualFrame.setField(field)* property are shape names, these data values are used instead of `StaticShapeFrame.shape`.

Parameter

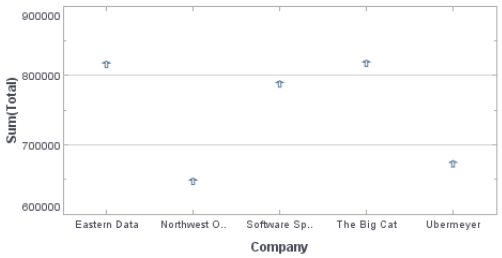
shape *GShape*, *GShape.ImageShape*, or *SVGShape* object

Note: Report script that modifies 'bindingInfo' should be placed in the onLoad handler.

Example (Report)

Bind a point-type chart to the sample ‘All Sales’ query, with ‘Company’ (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

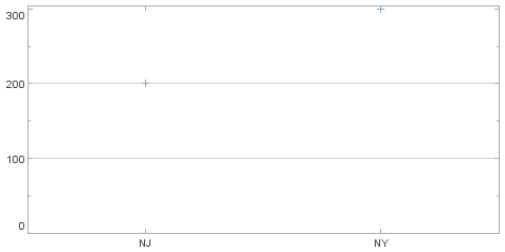
```
Chart1.bindingInfo.setShapeField("Company",Chart.STRING);
Chart1.bindingInfo.shapeFrame = new StaticShapeFrame;
Chart1.bindingInfo.shapeFrame.setShape(GShape.ARROWBAR);
```



Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ", 200], ["NY", 300]];
graph = new EGraph();
var frame = new StaticShapeFrame();
var elem = new PointElement("State", "Quantity");
frame.setShape(GShape.CROSS);
elem.setShapeFrame(frame);
graph.addElement(elem);
```



Inherits from

VisualFrame: setField, setLegendSpec, setScale, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.7.32 MultiShapeFrame

The MultiShapeFrame object contains the shape styles for visual chart objects on multiple dimensions.

Full Name

```
inetsoft.graph.aesthetic.MultiShapeFrame
```

MultiShapeFrame: setFields, setScales

Created by

VineShapeFrame
ThermoShapeFrame
StarShapeFrame
SunShapeFrame

BarShapeFrame
ProfileShapeFrame

Inherits from
VisualFrame: setField, setLegendSpec, setScale, setScaleOption

MultiShapeFrame.SETFIELDS(ARR)

Specifies the columns containing the shape specifications for the graphical elements.

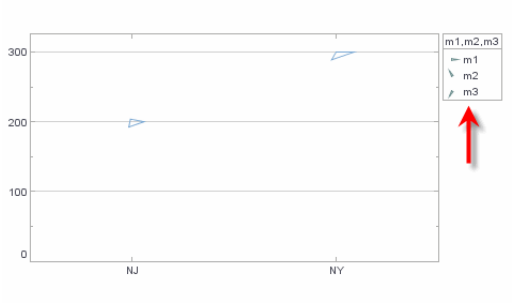
Parameter

arr	Array of Strings

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity", "m1", "m2", "m3"], ["NJ",
200, 6,3,4], ["NY", 300,8,2,5]];
graph = new EGraph();
var frame = new StarShapeFrame();
var elem = new PointElement("State", "Quantity");
frame.setFields(["m1", "m2", "m3"]);
elem.setShapeFrame(frame);
graph.addElement(elem);
```



Inherits from
VisualFrame: setField, setLegendSpec, setScale, setScaleOption

See Also
DataSet, to use a Data Block or query as the *dataset* source.

MultiShapeFrame.SETSCALES(ARR)

Specifies the scales to be used for each shape field.

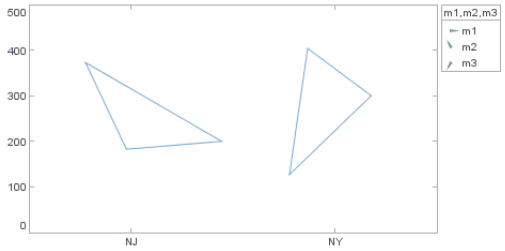
Parameter

arr	Array of *Scale*

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity", "m1", "m2", "m3"], ["NJ",
200, 50,50,5], ["NY", 300,20,30,50]];
graph = new EGraph();
var frame = new StarShapeFrame();
var elem = new PointElement("State", "Quantity");
var yscale = new LinearScale("Quantity");
yscale.setMax(500);
frame.setFields(["m1", "m2", "m3"]);
var scale1 = new LinearScale("m1");
var scale2 = new LinearScale("m2");
var scale3 = new LinearScale("m3");
scale1.setMax(10);
scale2.setMax(10);
scale3.setMax(10);
frame.setScales([scale1, scale2, scale3]);
elem.setShapeFrame(frame);
graph.setScale("Quantity",yscale)
graph.addElement(elem);
```



Inherits from

VisualFrame: setField, setLegendSpec, setScale, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.7.33 VineShapeFrame

The VineShapeFrame object contains the shape styles for three-dimensional “vine” elements. To create a VineShapeFrame object, call the VineShapeFrame constructor.

```
var frame = new VineShapeFrame("m1","m2","m3");
```

You can pass a set of field names (e.g., ‘m1’, ‘m2’, ‘m3’) to the constructor, or specify this later using the inherited *MultiShapeFrame.setFields(arr)* property. The dimensions are specified in the following order: `[angle, magnitude, radius]`.

- Angle: The angle of the stem line

- Magnitude: The length of the stem line
- Radius: The radius of the circle

Full Name

`inetsoft.graph.aesthetic.VineShapeFrame`

VineShapeFrame: setEndAngle, setStartAngle

Inherits from

MultiShapeFrame: setFields, setScales
VisualFrame: setField, setLegendSpec, setScale, setScaleOption

VineShapeFrame.SETENDANGLE(VALUE)

The angle to which the maximum angle in the data is mapped. (Note that the `max` property of any applied scale also affects the displayed angle.)

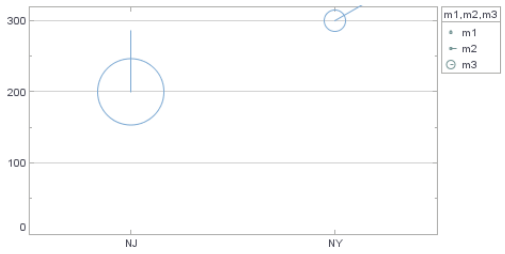
Parameter

`value` the angle in degrees

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity", "m1", "m2", "m3"],["NJ",
200,90,25,50],["NY", 300,30,15,15]];
graph = new EGraph();
var elem = new PointElement("State", "Quantity");
var frame = new VineShapeFrame();
var mscale = new LinearScale();
var rscale = new LinearScale();
mscale.setMin(0);
mscale.setMax(5);
rscale.setMin(0);
rscale.setMax(90);
frame.setScales([rscale, mscale, mscale]);
frame.setFields(["m1", "m2", "m3"]);
frame.setStartAngle(0);
frame.setEndAngle(90);
elem.setShapeFrame(frame);
graph.addElement(elem);
```



Inherits from
MultiShapeFrame: setFields, setScales
VisualFrame: setField, setLegendSpec, setScale, setScaleOption

See Also
DataSet, to use a Data Block or query as the *dataset* source.

VineShapeFrame.SETSTARTANGLE(VALUE)

The angle to which the minimum angle in the data is mapped. (Note that the `min` property of any applied scale also affects the displayed angle.)

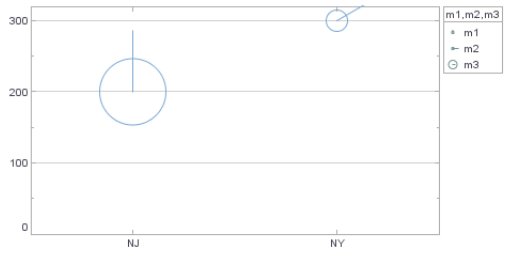
Parameter

value	the angle in degrees

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity", "m1", "m2", "m3"], ["NJ",
200,90,25,50], ["NY", 300,30,15,15]];
graph = new EGraph();
var elem = new PointElement("State", "Quantity");
var frame = new VineShapeFrame();
var mscale = new LinearScale();
var rscale = new LinearScale();
mscale.setMin(0);
mscale.setMax(5);
rscale.setMin(0);
rscale.setMax(90);
frame.setScales([rscale, mscale, mscale]);
frame.setFields(["m1", "m2", "m3"]);
frame.setStartAngle(0);
frame.setEndAngle(90);
elem.setShapeFrame(frame);
graph.addElement(elem);
```



Inherits from
MultiShapeFrame: setFields, setScales
VisualFrame: setField, setLegendSpec, setScale, setScaleOption

See Also
DataSet, to use a Data Block or query as the *dataset* source.

CR.7.34 ThermoShapeFrame

The ThermoShapeFrame object contains the shape styles for two-dimensional "thermometer" elements. To create a ThermoShapeFrame object, call the ThermoShapeFrame constructor.

```
var frame = new ThermoShapeFrame("Height", "Weight");
```

You can pass a pair of field names (e.g., 'Height', 'Weight') to the constructor, or specify this later using the inherited *MultiShapeFrame.setFields(arr)* property. The dimensions are specified in the order `[level, width]`.

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity","Height","Weight"], ["NJ",
200,50,1], ["NY", 300,30,4]];
graph = new EGraph();
var frame = new ThermoShapeFrame();
var elem = new PointElement("State", "Quantity");
var hscale = new LinearScale()
var wscale = new LinearScale()
hscale.setMin(0);
hscale.setMax(100);
wscale.setMin(0);
wscale.setMax(5);
frame.setFields(["Height", "Weight"]);
frame.setScales([hscale, wscale]);
elem.setShapeFrame(frame);
graph.addElement(elem);
```



Full Name

`inetsoft.graph.aesthetic.ThermoShapeFrame`

Inherits from

MultiShapeFrame: setFields, setScales
VisualFrame: setField, setLegendSpec, setScale, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.7.35 StarShapeFrame

The StarShapeFrame object contains the shape styles for multi-dimensional "star" (closed line) elements. To create a StarShapeFrame object, call the StarShapeFrame constructor.

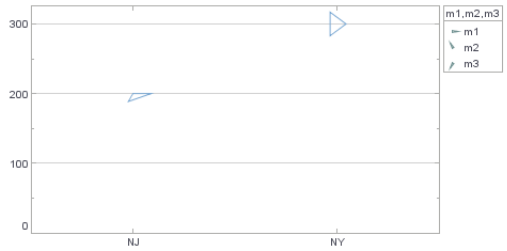
```
var frame = new StarShapeFrame("m1","m2","m3");
```

You can pass a set of field names (e.g., 'm1', 'm2', 'm3') to the constructor, or specify this later using the inherited *MultiShapeFrame.setFields(arr)* property.

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity","m1","m2","m3"], ["NJ",
200,5,1,3], ["NY", 300,3,4,4]];
graph = new EGraph();
var frame = new StarShapeFrame();
var elem = new PointElement("State", "Quantity");
frame.setFields(["m1","m2","m3"]);
elem.setShapeFrame(frame);
graph.addElement(elem);
```



Full Name

`inetsoft.graph.aesthetic.StarShapeFrame`

Inherits from

MultiShapeFrame: setFields, setScales
VisualFrame: setField, setLegendSpec, setScale, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.7.36 SunShapeFrame

The SunShapeFrame object contains the shape styles for multidimensional "sun" (radial line) elements. To create a SunShapeFrame object, call the SunShapeFrame constructor.

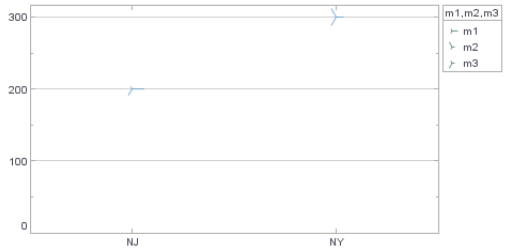
```
var frame = new SunShapeFrame("m1","m2","m3");
```

You can pass a set of field names (e.g., ‘m1’, ‘m2’, ‘m3’) to the constructor, or specify this later using the inherited *MultiShapeFrame.setFields(arr)* property.

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity","m1","m2","m3"], ["NJ",
200,5,1,3], ["NY", 300,3,4,4]];
graph = new EGraph();
var frame = new SunShapeFrame();
var elem = new PointElement("State", "Quantity");
frame.setFields(["m1","m2","m3"]);
elem.setShapeFrame(frame);
graph.addElement(elem);
```



Full Name

```
inetsoft.graph.aesthetic.SunShapeFrame
```

Inherits from

MultiShapeFrame: setFields, setScales
VisualFrame: setField, setLegendSpec, setScale, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.7.37 BarShapeFrame

The BarShapeFrame object contains the shape styles for multi-dimensional “mini-bar chart” elements. To create a BarShapeFrame object, call the BarShapeFrame constructor.

```
var frame = new BarShapeFrame("m1","m2","m3");
```

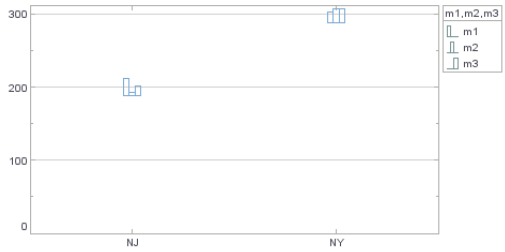
You can pass a set of field names (e.g., ‘m1’, ‘m2’, ‘m3’) to the constructor, or specify this later using the inherited *MultiShapeFrame.setFields(arr)* property. The dimensions are mapped to the bar heights from left to right.

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and

Adding Component Script in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity","m1","m2","m3"], ["NJ",
200,5,1,3], ["NY", 300,3,4,4]];
graph = new EGraph();
var frame = new BarShapeFrame();
var elem = new PointElement("State", "Quantity");
frame.setFields(["m1","m2","m3"]);
elem.setShapeFrame(frame);
graph.addElement(elem);
```



Full Name

`inetsoft.graph.aesthetic.BarShapeFrame`

Inherits from

MultiShapeFrame: setFields, setScales
VisualFrame: setField, setLegendSpec, setScale, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.7.38 PieShapeFrame

The PieShapeFrame object contains the shape styles for multi-dimensional "mini-pie" elements to be used together with a point-type (`PointElement`) chart. To create a PieShapeFrame object, call the PieShapeFrame constructor.

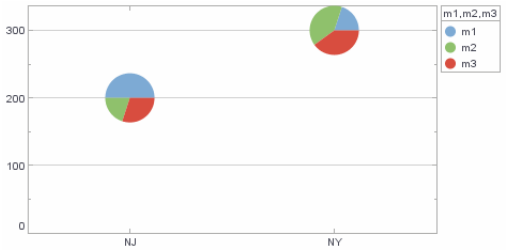
```
var frame = new PieShapeFrame("m1","m2","m3");
```

You can pass a set of field names (e.g., 'm1', 'm2', 'm3') to the constructor, or specify this later using the inherited *MultiShapeFrame.setFields(arr)* property. The dimension values are mapped proportionately to the areas of the pie slices.

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity","m1","m2","m3"], ["NJ",
200,5,2,3], ["NY", 300,2,4,4]];
graph = new EGraph();
var frame = new PieShapeFrame();
var elem = new PointElement("State", "Quantity");
var sframe = new StaticSizeFrame(15);
frame.setFields(["m1","m2","m3"]);
elem.setShapeFrame(frame);
elem.setSizeFrame(sframe);
graph.addElement(elem);
```



Full Name

```
inetsoft.graph.aesthetic.PieShapeFrame
```

Inherits from

MultiShapeFrame: setFields, setScales
VisualFrame: setField, setLegendSpec, setScale, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.7.39 ProfileShapeFrame

The ProfileShapeFrame object contains the shape styles for multi-dimensional "mini-line chart" elements. To create a ProfileShapeFrame object, call the ProfileShapeFrame constructor.

```
var frame = new ProfileShapeFrame("m1","m2","m3");
```

You can pass a set of field names (e.g., 'm1', 'm2', 'm3') to the constructor, or specify this later using the inherited *MultiShapeFrame.setFields(arr)* property. The dimensions are mapped to the line segments from left to right.

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity","m1","m2","m3"], ["NJ",
200,5,1,3], ["NY", 300,3,4,4]];
graph = new EGraph();
var frame = new ProfileShapeFrame();
var elem = new PointElement("State", "Quantity");
```

```
frame.setFields(["m1","m2","m3"]);
elem.setShapeFrame(frame);
graph.addElement(elem);
```



Full Name

`inetsoft.graph.aesthetic.ProfileShapeFrame`

Inherits from

MultiShapeFrame: setFields, setScales
VisualFrame: setField, setLegendSpec, setScale, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.7.40 TextFrame

The TextFrame object contains a mapping between values and displayed text.

Full Name

`inetsoft.graph.aesthetic.TextFrame`

TextFrame: setText

Inherits from

VisualFrame: setField, setLegendSpec, setScale, setScaleOption

Created by

DefaultTextFrame

TextFrame.SETTEXT(VALUE,TEXT)

Specifies the mapping between a value and the text that should be displayed.

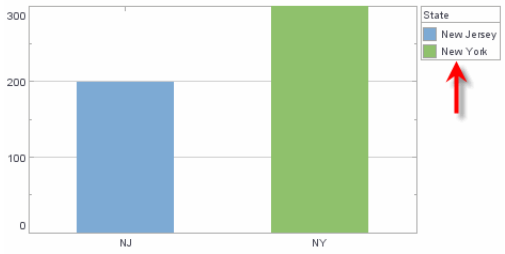
Parameters

value	the data value to be replaced
text	the replacement string

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ",200], ["NY",300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var cframe = new CategoricalColorFrame();
var tframe = new DefaultTextFrame();
var spec = cframe.getLegendSpec();
cframe.setField("State");
tframe.setText('NJ','New Jersey');
tframe.setText('NY','New York');
spec.setTextFrame(tframe)
elem.setColorFrame(cframe);
graph.addElement(elem);
```



Inherits from

VisualFrame: setField, setLegendSpec, setScale, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.7.41 DefaultTextFrame

The DefaultTextFrame object contains data to display in text form, as well as a mapping between these values and any desired replacement text. To create a DefaultTextFrame object, call the DefaultTextFrame constructor.

```
var frame = new DefaultTextFrame('Quantity');
```

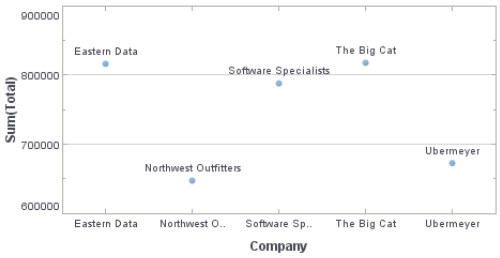
You can pass a field name (e.g., 'Quantity') to the constructor, or specify this later using the inherited *VisualFrame.setField(field)* property. To stylize or reposition text created using a DefaultTextFrame, use the *GraphElement.setTextSpec(spec)* and *GraphElement.setLabelPlacement(value)* options.

```
Note: Report script that modifies 'bindingInfo' should be placed in
the onLoad handler.
```

Example (Report)

Bind a point-type chart to the sample 'All Sales' query, with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

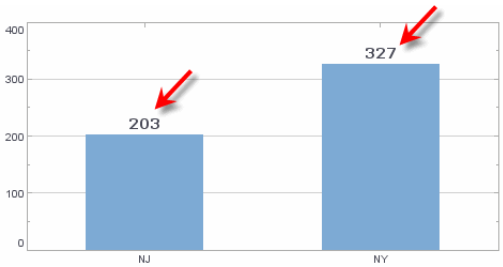
```
Chart1.bindingInfo.setTextField("Company",Chart.STRING)
Chart1.bindingInfo.textFrame = new DefaultTextFrame;
```



Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

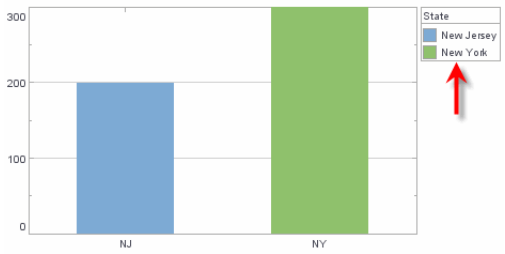
Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ",203], ["NY",327]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var spec = new TextSpec();
var frame = new DefaultTextFrame();
spec.setFont(java.awt.Font('Verdana',java.awt.Font.BOLD,
14));
frame.setField("Quantity");
elem.setTextFrame(frame);
elem.setTextSpec(spec);
graph.addElement(elem);
```



Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ",200], ["NY",300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var cframe = new CategoricalColorFrame();
var tframe = new DefaultTextFrame();
var spec = cframe.getLegendSpec();
cframe.setField("State");
tframe.setText('NJ','New Jersey');
tframe.setText('NY','New York');
spec.setTextFrame(tframe)
elem.setColorFrame(cframe);
graph.addElement(elem);
```



Full Name

`inetsoft.graph.aesthetic.DefaultTextFrame`

Inherits from

TextFrame: setText
VisualFrame: setField, setLegendSpec, setScale, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.7.42 LineFrame

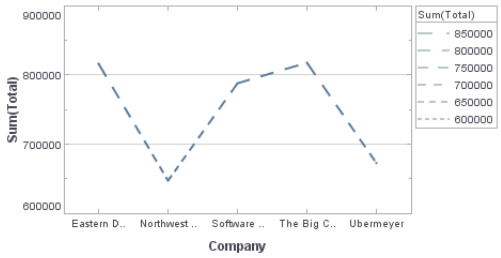
The LineFrame object contains the line design for visual chart objects. LineFrame objects allow you to represent additional data dimensions by using the physical attributes of chart elements, or to apply a fixed (static) line style.

`Note: Report script that modifies 'bindingInfo' should be placed in the onLoad handler.`

Example (Report)

Bind a line-type chart to the sample 'All Sales' query, with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

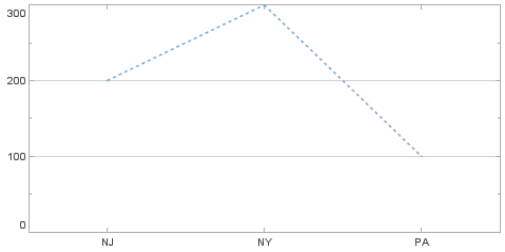
```
Chart1.bindingInfo.setShapeField("Total",Chart.NUMBER);
Chart1.bindingInfo.lineFrame = new LinearLineFrame;
```



`Note: Script that modifies 'graph' should be placed at the element level. See` *`Adding Element-Level Script`* `in` *`Report Scripting`* `and` *`Adding Component Script`* `in` *`Dashboard Scripting`* `for more information.`

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ",200], ["NY",300],
["PA",100]];
graph = new EGraph();
var elem = new LineElement("State", "Quantity");
var frame = new StaticLineFrame();
frame.setLine(GLine.DASH_LINE);
elem.setLineFrame(frame);
graph.addElement(elem)
```



Full Name

```
inetsoft.graph.aesthetic.LineFrame
```

Created by

LinearLineFrame
CategoricalLineFrame
StaticLineFrame

Inherits from

VisualFrame: setField, setLegendSpec, setScale, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.7.43 StaticLineFrame

The StaticLineFrame object contains a line frame defined by explicit line data in the *VisualFrame.setField(field)* column, or by the fixed line in *StaticLineFrame.setLine(value)*. To create a StaticLineFrame object, call the StaticLineFrame constructor.

```
var frame = new StaticLineFrame(GLine.LARGE_DASH);
```

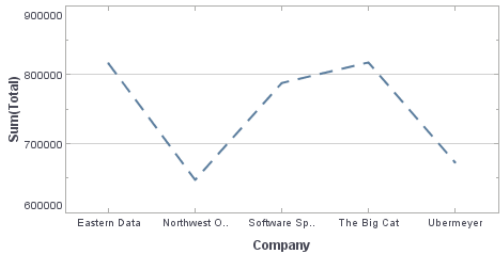
You can pass a *GLine* object or a field name (e.g., 'Line') to the constructor, or specify this later using the *StaticLineFrame.setLine(value)* or inherited *VisualFrame.setField(field)* property.

```
Note: Report script that modifies 'bindingInfo' should be placed in
the onLoad handler.
```

Example (Report)

Bind a line-type chart to the sample 'All Sales' query, with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

```
Chart1.bindingInfo.lineFrame = new StaticLineFrame;
Chart1.bindingInfo.lineFrame.line = GLine.LARGE_DASH;
```



Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State","Quantity","Line"],
["NJ",200,Chart.THICK_LINE], ["NY",300,Chart.MEDIUM_DASH]];
graph = new EGraph();
var elem = new PointElement("State", "Quantity");
var sizeframe = new StaticSizeFrame(9);
var lineframe = new StaticLineFrame();
lineframe.setField("Line");
elem.setLineFrame(lineframe);
elem.setSizeFrame(sizeframe);
graph.addElement(elem);
```



Full Name

```
inetsoft.graph.aesthetic.StaticLineFrame
```

StaticLineFrame: setLine

Inherits from

VisualFrame: setField, setLegendSpec, setScale, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

StaticLineFrame.SETLINE(VALUE)

Specifies the static line style for graphical elements. If the data in the column assigned to the inherited *VisualFrame.setField(field)* property are *GLine* objects, these data values are used instead of `StaticLineFrame.line`.

Parameter

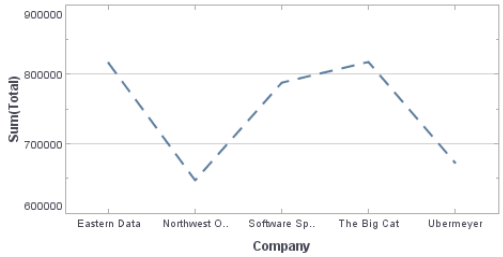
value *GLine* or
`Chart.NONE  // Empty border`

`Note: Report script that modifies 'bindingInfo' should be placed in the onLoad handler.`

Example (Report)

Bind a line-type chart to the sample 'All Sales' query, with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

```
Chart1.bindingInfo.setShapeField("Company",Chart.NUMBER)
Chart1.bindingInfo.lineFrame = new StaticLineFrame;
Chart1.bindingInfo.lineFrame.line = GLine.LARGE_DASH;
```



`Note: Script that modifies 'graph' should be placed at the element level. See` *`Adding Element-Level Script`* `in` *`Report Scripting`* `and` *`Adding Component Script`* `in` *`Dashboard Scripting`* `for more information.`

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ", 300], ["NY", 200]];
graph = new EGraph();
var elem = new PointElement("State", "Quantity")
var frame = new StaticLineFrame();
frame.setLine(GLine.DOT_LINE);
elem.setLineFrame(frame);
graph.addElement(elem);
```



Inherits from

VisualFrame: setField, setLegendSpec, setScale, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.7.44 LinearLineFrame

The LinearLineFrame object contains a continuous line frame that returns varying line styles. To create a LinearLineFrame object, call the LinearLineFrame constructor.

```
var frame = new LinearLineFrame('Quantity');
```

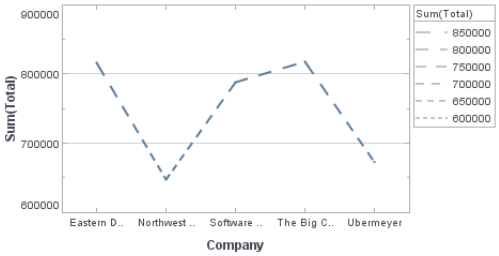
You can pass the name of a field (e.g., 'Quantity') to the constructor, or specify this later using the inherited *VisualFrame.setField(field)* property.

Note: Report script that modifies 'bindingInfo' should be placed in the onLoad handler.

Example (Report)

Bind a line-type chart to the sample 'All Sales' query, with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

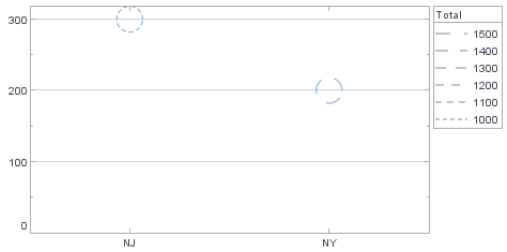
```
Chart1.bindingInfo.setShapeField("Total",Chart.NUMBER);
Chart1.bindingInfo.lineFrame = new LinearLineFrame;
```



Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity","Total"], ["NJ",300,1000],
["NY",200,1500]];
graph = new EGraph();
var elem = new PointElement("State", "Quantity")
var lframe = new LinearLineFrame();
var sframe = new StaticSizeFrame();
lframe.setField("Total");
sframe.setSize(10);
elem.setLineFrame(lframe);
elem.setSizeFrame(sframe);
graph.addElement(elem);
```



Full Name

```
inetsoft.graph.aesthetic.LinearLineFrame
```

Inherits from

VisualFrame: setField, setLegendSpec, setScale, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.7.45 CategoricalLineFrame

The CategoricalLineFrame object contains a unique line style for each discrete value. To create a CategoricalLineFrame object, call the CategoricalLineFrame constructor.

```
var frame = new CategoricalLineFrame('Quantity');
```

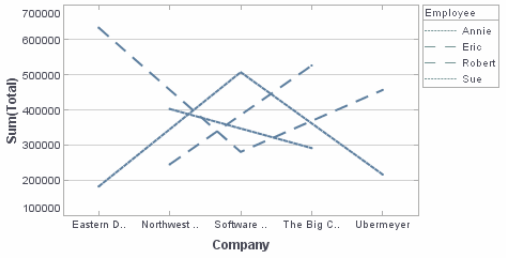
You can pass the name of a field (e.g., 'Quantity') to the constructor, or specify this later using the inherited *VisualFrame.setField(field)* property.

```
Note: Report script that modifies 'bindingInfo' should be placed in
the onLoad handler.
```

Example (Report)

Bind a line-type chart to the sample 'All Sales' query, with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

```
Chart1.bindingInfo.setShapeField("Employee",Chart.STRING);
Chart1.bindingInfo.lineFrame = new CategoricalLineFrame;
Chart1.bindingInfo.lineFrame.setLine('Robert',
GLine.LARGE_DASH);
Chart1.bindingInfo.lineFrame.setLine('Eric',
GLine.LARGE_DASH);
Chart1.bindingInfo.lineFrame.setLine('Sue',GLine.DOT_LINE);
Chart1.bindingInfo.lineFrame.setLine('Annie',
GLine.DOT_LINE);
```



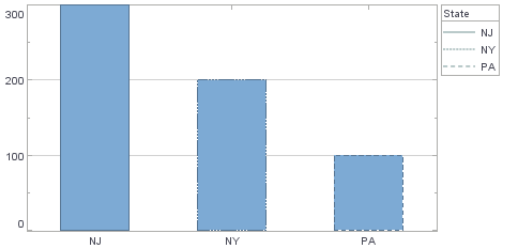
Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State",
"Quantity"],["NJ",300],["NY",200],["PA",100]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity")
var frame = new CategoricalLineFrame();
frame.setField("State");
elem.setLineFrame(frame);
graph.addElement(elem);
```

Full Name

inetsoft.graph.aesthetic.CategoricalLineFrame



Inherits from

VisualFrame: setField, setLegendSpec, setScale, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

CategoricalLineFrame: setLine

CategoricalLineFrame.SETLINE(VAL,LINE)

Assigns a line style to the specified value.

Parameter

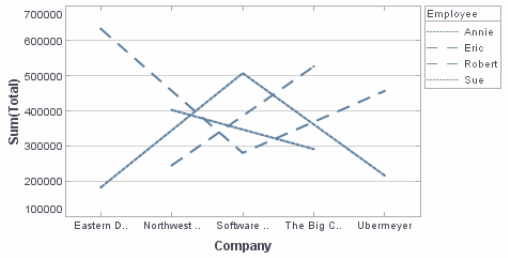
val	a data value
line	a *GLine* value

Note: Report script that modifies 'bindingInfo' should be placed in the onLoad handler.

Example (Report)

Bind a line-type chart to the sample 'All Sales' query, with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

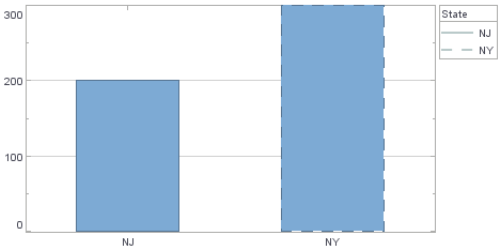
```
Chart1.bindingInfo.setShapeField("Employee",Chart.STRING);
Chart1.bindingInfo.lineFrame = new CategoricalLineFrame;
Chart1.bindingInfo.lineFrame.setLine('Robert',
GLine.LARGE_DASH);
Chart1.bindingInfo.lineFrame.setLine('Eric',
GLine.LARGE_DASH);
Chart1.bindingInfo.lineFrame.setLine('Sue',
GLine.DOT_LINE);
Chart1.bindingInfo.lineFrame.setLine('Annie',
GLine.DOT_LINE);
```



Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State","Quantity"], ["NJ",200], ["NY",300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity")
frame = new CategoricalLineFrame();
frame.setField("State");
frame.setLine('NJ',GLine.THIN_LINE);
frame.setLine('NY',GLine.LARGE_DASH);
elem.setLineFrame(frame);
graph.addElement(elem);
```



Inherits from

VisualFrame: setField, setLegendSpec, setScale, setScaleOption

See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.8 Chart Annotation and Decoration

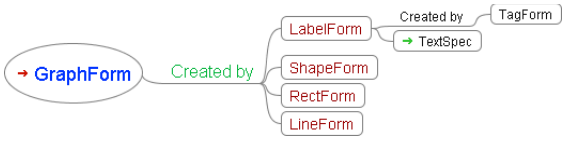
This section describes objects used to add arbitrary text, shapes, and lines to a chart.

CR.8.1 GraphForm

The GraphForm object contains information for form (i.e., shape) elements manually drawn on the chart.

Full Name

```
inetsoft.graph.guide.form.GraphForm
```



GraphForm: setColor, setFill, setInsidePlot, setLine, setMeasure, setZIndex

Created by
LabelForm
TagForm
ShapeForm
RectForm
LineForm

GraphForm.SETCOLOR(VALUE)

Specifies the line and fill color of the form.

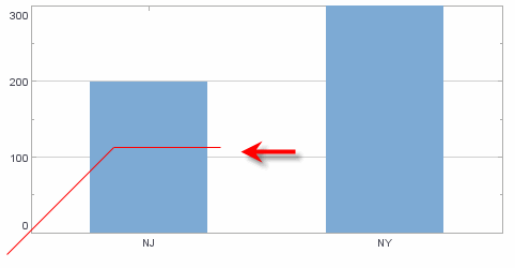
Type

```
value        Number
```

```
Note: Script that modifies 'graph' should be placed at the element
level. See Adding Element-Level Script in Report Scripting and
Adding Component Script in Dashboard Scripting for more
information.
```

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ", 200], ["NY", 300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var form = new LineForm();
form.addPoint(new java.awt.Point(0,0));
form.addPoint(new java.awt.Point(100,100));
form.addPoint(new java.awt.Point(200,100));
form.setColor(java.awt.Color(0xff0000));
graph.addForm(form);
graph.addElement(elem);
```



See Also

DataSet, to use a Data Block or query as the *dataset* source.

GraphForm.SETFILL(BOOLEAN)

Specifies whether the form should be filled or unfilled.

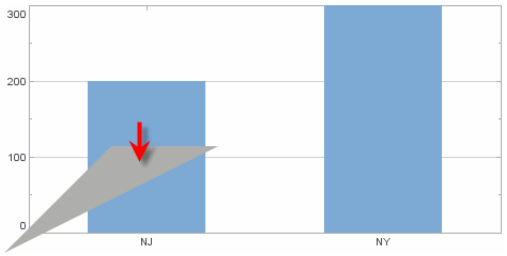
Type

boolean true: fill the shape
false: do not fill the shape

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ", 200], ["NY", 300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var form = new LineForm();
form.addPoint(new java.awt.Point(0,0));
form.addPoint(new java.awt.Point(100,100));
form.addPoint(new java.awt.Point(200,100));
form.setFill(true);
graph.addForm(form);
graph.addElement(elem);
```



See Also

DataSet, to use a Data Block or query as the *dataset* source.

GraphForm.SETINPLOT(BOOLEAN)

For forms that use relative positioning (values or tuples), specifies whether the chart should be resized so that forms remain fully visible in the chart area.

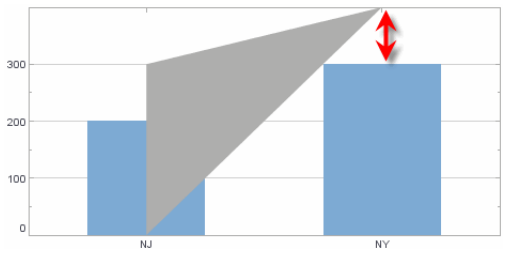
Type

boolean	true: resize chart false: do not resize chart (crop forms)

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ", 200], ["NY", 300]];
graph = new EGraph();
var form = new LineForm();
var elem = new IntervalElement("State", "Quantity");
form.addValues(['NJ',0]);
form.addValues(['NJ',300]);
form.addValues(['NY',400]);
form.setInPlot(true);
form.setFill(true);
graph.addForm(form);
graph.addElement(elem);
```



See Also

AxisSpec.setInPlot(boolean), to adjust plot boundaries to include graph labels.

GraphElement.setInPlot(boolean), to adjust plot boundaries to include graph elements.

DataSet, to use a Data Block or query as the *dataset* source.

GraphForm.SETLINE(VALUE)

Specifies the line style used to draw the form.

Type

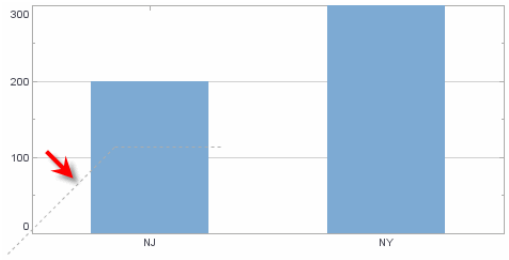
value	a *GLine* constant

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and

Adding Component Script in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ", 200], ["NY", 300]];
graph = new EGraph();
var form = new LineForm();
var elem = new IntervalElement("State", "Quantity");
form.addPoint(new java.awt.Point(0,0));
form.addPoint(new java.awt.Point(100,100));
form.addPoint(new java.awt.Point(200,100));
form.setLine(Chart.DASH_LINE);
graph.addForm(form);
graph.addElement(elem);
```



See Also

DataSet, to use a Data Block or query as the *dataset* source.

GraphForm.SETMEASURE(COL)

Specifies the measure for which the form should be displayed. If the chart contains a measure of this name, the form object is displayed. Otherwise it is not displayed. This is useful for *FacetCoord* charts, when the form should be displayed only for the chart that represents a particular measure.

Type

col name of a column (String)

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity", "Sales"], ["NJ", 200, 800],
["NY", 300, 600]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var elem2 = new IntervalElement("State", "Sales");
var form = new LabelForm();
form.setValues(['NJ',200]);
form.setLabel("NJ Sales");
form.setAlignmentX(Chart.CENTER_ALIGNMENT);
form.setMeasure("Sales");
graph.addForm(form);
var scale = new CategoricalScale("State");
var qscale = new LinearScale("Quantity");
```

```
var sscale = new LinearScale("Sales");
var qcoord = new RectCoord(scale, qscale);
var scoord = new RectCoord(scale, sscale);
var facet = new FacetCoord();
facet.setInnerCoordinates([qcoord, scoord]);
graph.setCoordinate(facet);
graph.addElement(elem);
graph.addElement(elem2);
```



See Also

DataSet, to use a Data Block or query as the *dataset* source.

GraphForm.SETXOFFSET(VALUE)

Specifies an offset (in pixels) by which to shift the form object horizontally. Positive values offset the object to the right; negative values offset the object to the left.

Type

value an integer (pixels)

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"],["NJ", 200],["NY", 300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var form = new LabelForm();
form.setLabel("label1");
form.setValues(['NY', 100]);
form.setXOffset(-50)
form.setYOffset(100)
graph.addForm(form);
graph.addElement(elem);
```

See Also

DataSet, to use a Data Block or query as the *dataset* source.

GraphForm.SETYOFFSET(VALUE)

Specifies an offset (in pixels) by which to shift the form object vertically. Positive values offset the object upward; negative values offset the object downward.

Type

value	an integer (pixels)

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"],["NJ", 200],["NY", 300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var form = new LabelForm();
form.setLabel("label1");
form.setValues(['NY', 100]);
form.setXOffset(-50)
form.setYOffset(100)
graph.addForm(form);
graph.addElement(elem);
```

See Also

DataSet, to use a Data Block or query as the *dataset* source.

GraphForm.SETZINDEX(VALUE)

Specifies the layering order for forms. A shape with a larger zIndex overlays a shape with a smaller zIndex. To ensure that a manually drawn shape appears in front of other chart elements, select a large zIndex. See *Z-Index Defaults* for the values of standard chart objects.

Type

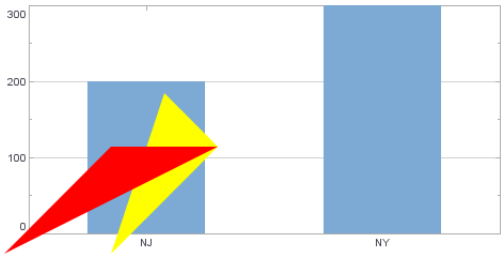
value	the index, a positive integer

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State","Quantity"], ["NJ",200], ["NY",300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var form1 = new LineForm();
var form2 = new LineForm();
form1.addPoint(new java.awt.Point(0,0));
form1.addPoint(new java.awt.Point(100,100));
form1.addPoint(new java.awt.Point(200,100));
form1.setColor(java.awt.Color(0xff0000));
form1.setFill(true);
form1.setZIndex(300);
form2.addPoint(new java.awt.Point(100,0));
form2.addPoint(new java.awt.Point(150,150));
form2.addPoint(new java.awt.Point(200,100));
form2.setColor(java.awt.Color(0xffff00));
form2.setFill(true);
form2.setZIndex(200);
graph.addForm(form1);
graph.addForm(form2);
```

```
graph.addElement(elem);
```



See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.8.2 LineForm

The LineForm object contains information for lines manually drawn on the chart. To create a LineForm object, call the LineForm constructor.

```
var line = new LineForm();
```

Full Name

```
inetsoft.graph.guide.form.LineForm
```

LineForm: setEndArrow, setPoints, setStartArrow, setTuples, setValues

Inherits from

GraphForm: setColor, setFill, setInsidePlot, setLine, setMeasure, setZIndex

LineForm.SETENDARROW(BOOLEAN)

Specifies whether an arrow should be drawn at the end of the line (i.e., at the last point specified for the line form).

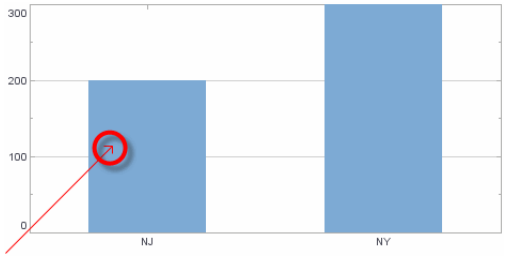
Type

boolean true: draw arrow
false: no arrow

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ", 200], ["NY", 300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var form = new LineForm();
form.addPoint(new java.awt.Point(0,0));
form.addPoint(new java.awt.Point(100,100));
form.setEndArrow(true);
form.setColor(java.awt.Color(0xff0000));
graph.addForm(form);
graph.addElement(elem);
```



See Also

DataSet, to use a Data Block or query as the *dataset* source.

Inherits from

GraphForm: setColor, setFill, setInsidePlot, setLine, setMeasure, setZIndex

LineForm.ADDPOINT(VALUE)

Specifies the pixel location (integer values) or proportional location (fractional values) defining a point on the line. (Positive values specify distance from left/bottom. Negative values specify distance from right/top.)

Type

value	a subclass of java.awt.geom.Point2D e.g., java.awt.Point for pixels java.awt.geom.Point2D.Double for proportion

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

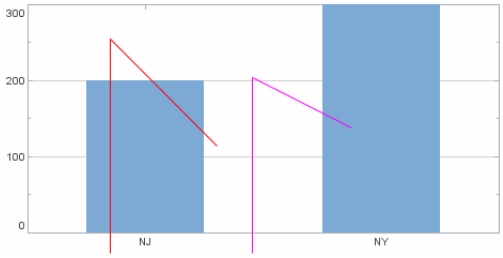
Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ", 200], ["NY", 300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var form1 = new LineForm();
var form2 = new LineForm();

// Set form1 points in pixels:
form1.addPoint(new java.awt.Point(100, 0));
form1.addPoint(new java.awt.Point(100, 200));
form1.addPoint(new java.awt.Point(200,100));
form1.setColor(java.awt.Color(0xff0000));

// Set form2 points proportionally:
form2.addPoint(new java.awt.geom.Point2D.Double(.5,0));
form2.addPoint(new java.awt.geom.Point2D.Double(.5,.7));
form2.addPoint(new java.awt.geom.Point2D.Double(.7,.5));
form2.setColor(java.awt.Color(0xff00ff));

graph.addForm(form1);
graph.addForm(form2);
graph.addElement(elem);
```



Inherits from

GraphForm: setColor, setFill, setInsidePlot, setLine, setMeasure, setZIndex

See Also

DataSet, to use a Data Block or query as the *dataset* source.

LineForm.SETSTARTARROW(BOOLEAN)

Specifies whether an arrow should be drawn at the start of the line (i.e., at the first point specified for the line form).

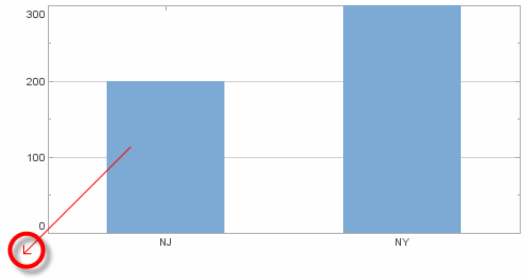
Type

boolean	true: draw arrow false: no arrow

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ", 200], ["NY", 300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var form = new LineForm();
form.addPoint(new java.awt.Point(0,0));
form.addPoint(new java.awt.Point(100,100));
form.setStartArrow(true);
form.setColor(java.awt.Color(0xff0000));
graph.addForm(form);
graph.addElement(elem);
```



Inherits from

GraphForm: setColor, setFill, setInsidePlot, setLine, setMeasure, setZIndex

See Also

DataSet, to use a Data Block or query as the *dataset* source.

LineForm.ADDTUPLE(VALUE)

Specifies a point defining the line in logical space. The coordinates of the points are relative to the prevailing axis scaling.

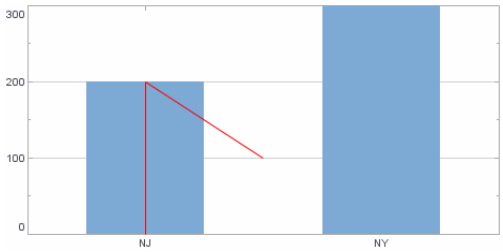
Type

value	An [X,Y] pair

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ", 200], ["NY", 300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var form = new LineForm();
form.addTuple([.5, 0]);
form.addTuple([.5, 200]);
form.addTuple([1,100]);
form.setColor(java.awt.Color(0xff0000));
graph.addForm(form);
graph.addElement(elem);
```



Inherits from

GraphForm: setColor, setFill, setInsidePlot, setLine, setMeasure, setZIndex

See Also

DataSet, to use a Data Block or query as the *dataset* source.

LineForm.ADDVALUES(VALUE)

Specifies a point defining the line in logical space. The coordinates of the points are relative to prevailing axis scaling prior to transformation. So, for

a categorical X-axis (e.g., 'NJ', 'NY', 'PA', etc.), the X-values of points should specify a categorical value (e.g., 'NJ').

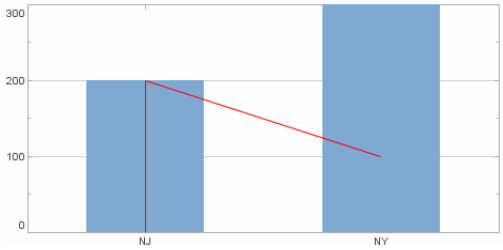
Type

value an [X,Y] pair

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ", 200], ["NY", 300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var form = new LineForm();
form.addValues(['NJ', 0]);
form.addValues(['NJ', 200]);
form.addValues(['NY', 100]);
form.setColor(java.awt.Color(0xff0000));
graph.addForm(form);
graph.addElement(elem);
```



Inherits from

GraphForm: setColor, setFill, setInsidePlot, setLine, setMeasure, setZIndex

See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.8.3 RectForm

The RectForm object contains information for rectangles manually drawn on the chart. To create a RectForm object, call the RectForm constructor.

```
var rect = new RectForm();
```

Full Name

```
inetsoft.graph.guide.form.RectForm
```

RectForm: setBottomRightPoint, setBottomRightTuple, setBottomRightValues, setTopLeftPoint, setTopLeftTuple, setTopLeftValues

Inherits from

GraphForm: setColor, setFill, setInsidePlot, setLine, setMeasure, setZIndex

RectForm.SETBOTTOMRIGHTPOINT(VALUE)

Specifies the pixel location (integer values) or proportional location (fractional values) defining the bottom right corner of the rectangle. (Positive values specify distance from left/bottom. Negative values specify distance from right/top.)

Type

value	a subclass of java.awt.geom.Point2D e.g., java.awt.Point for pixels java.awt.geom.Point2D.Double for proportion

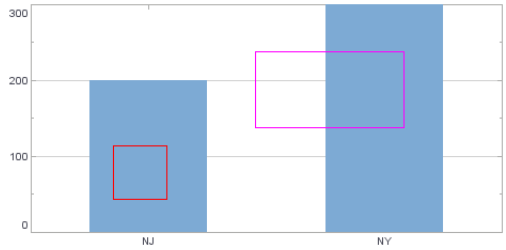
Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ", 200], ["NY", 300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var rect1 = new RectForm();
var rect2 = new RectForm();

// Set rect1 points in pixels:
rect1.setTopLeftPoint(new java.awt.Point(100, 100));
rect1.setBottomRightPoint(new java.awt.Point(150, 50));
rect1.setColor(java.awt.Color(0xff0000));

// Set rect2 points proportionally:
rect2.setTopLeftPoint(new
java.awt.geom.Point2D.Double(.5,.8));
rect2.setBottomRightPoint(new
java.awt.geom.Point2D.Double(.8,.5));
rect2.setColor(java.awt.Color(0xff00ff));
graph.addForm(rect1);
graph.addForm(rect2);
graph.addElement(elem);
```



Inherits from

GraphForm: setColor, setFill, setInsidePlot, setLine, setMeasure, setZIndex

See Also

DataSet, to use a Data Block or query as the *dataset* source.

RectForm.SETBOTTOMRIGHTTUPLE(VALUE)

Specifies the point in logical space for the bottom right corner of the rectangle. The coordinates of the point are relative to the prevailing axis scaling.

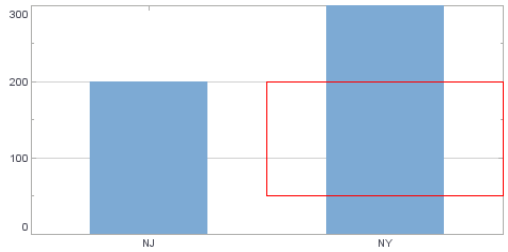
Type

value An [X,Y] pair

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ", 200], ["NY", 300]];
graph = new EGraph();
var rect = new RectForm();
rect.setTopLeftTuple([1,200]);
rect.setBottomRightTuple([2,50]);
rect.setColor(java.awt.Color(0xff0000));
graph.addForm(rect);
graph.addElement(new IntervalElement("State", "Quantity"));
```



Inherits from

GraphForm: setColor, setFill, setInsidePlot, setLine, setMeasure, setZIndex

See Also

DataSet, to use a Data Block or query as the *dataset* source.

RectForm.SETBOTTOMRIGHTVALUES(VALUE)

Specifies the point in logical space for the bottom right corner of the rectangle. The coordinates of the point are relative to prevailing axis scaling prior to transformation. So, for a categorical X-axis (e.g., 'NJ', 'NY', 'PA', etc.), the X-value of bottomRightValues should specify a categorical value (e.g., 'NJ').

Type

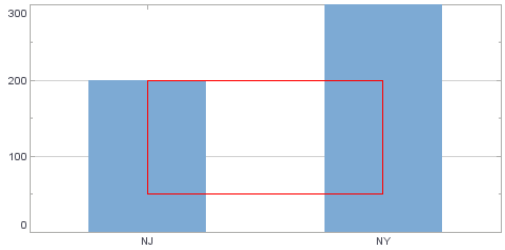
value an [X,Y] pair

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and

Adding Component Script in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ", 200], ["NY", 300]];
graph = new EGraph();
var rect = new RectForm();
rect.setTopLeftValues(['NJ',200]);
rect.setBottomRightValues(['NY',50]);
rect.setColor(java.awt.Color(0xff0000));
graph.addForm(rect);
graph.addElement(new IntervalElement("State", "Quantity"));
```



Inherits from

GraphForm: setColor, setFill, setInsidePlot, setLine, setMeasure, setZIndex

See Also

DataSet, to use a Data Block or query as the *dataset* source.

RectForm.SETTOPLEFTPOINT(VALUE)

Specifies the pixel location (integer values) or proportional location (fractional values) defining the top left corner of the rectangle. (Positive values specify distance from left/bottom. Negative values specify distance from right/top.)

Type

value	a subclass of java.awt.geom.Point2D e.g., java.awt.Point for pixels java.awt.geom.Point2D.Double for proportion

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

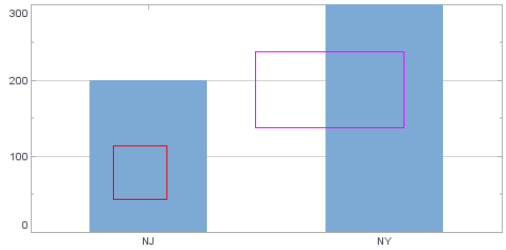
Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ", 200], ["NY", 300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var rect1 = new RectForm();
var rect2 = new RectForm();

// Set rect1 points in pixels:
```

```
rect1.setTopLeftPoint(new java.awt.Point(100, 100));
rect1.setBottomRightPoint(new java.awt.Point(150, 50));
rect1.setColor(java.awt.Color(0xff0000));

// Set rect2 points proportionally:
rect2.setTopLeftPoint(new
java.awt.geom.Point2D.Double(.5,.8));
rect2.setBottomRightPoint(new
java.awt.geom.Point2D.Double(.8,.5));
rect2.setColor(java.awt.Color(0xff00ff));
graph.addForm(rect1);
graph.addForm(rect2);
graph.addElement(elem);
```



Inherits from

GraphForm: setColor, setFill, setInsidePlot, setLine, setMeasure, setZIndex

See Also

DataSet, to use a Data Block or query as the *dataset* source.

RectForm.SETTOPLEFTTUPLE(VALUE)

Specifies the point in logical space for the top left corner of the rectangle. The coordinates of the point are relative to the prevailing axis scaling.

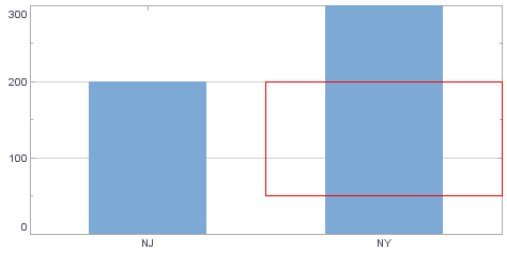
Type

value An [X,Y] pair

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ", 200], ["NY", 300]];
graph = new EGraph();
var rect = new RectForm();
rect.setTopLeftTuple([1,200]);
rect.setBottomRightTuple([2,50]);
rect.setColor(java.awt.Color(0xff0000));
graph.addForm(rect);
graph.addElement(new IntervalElement("State", "Quantity"));
```



Inherits from
GraphForm: setColor, setFill, setInsidePlot, setLine, setMeasure, setZIndex

See Also
DataSet, to use a Data Block or query as the *dataset* source.

RectForm.SETTOPLEFTVALUES(VALUE)

Specifies the point in logical space for the top left corner of the rectangle. The coordinates of the point are relative to prevailing axis scaling prior to transformation. So, for a categorical X-axis (e.g., 'NJ', 'NY', 'PA', etc.), the X-value of topLeftValues should specify a categorical value (e.g., 'NJ').

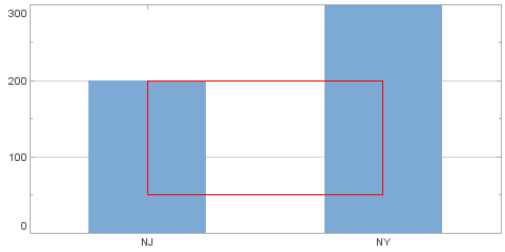
Type

value	an [X,Y] pair

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ", 200], ["NY", 300]];
graph = new EGraph();
var rect = new RectForm();
rect.setTopLeftValues(['NJ',200]);
rect.setBottomRightValues(['NY',50]);
rect.setColor(java.awt.Color(0xff0000));
graph.addForm(rect);
graph.addElement(new IntervalElement("State", "Quantity"));
```



Inherits from
GraphForm: setColor, setFill, setInsidePlot, setLine, setMeasure, setZIndex

See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.8.4 LabelForm

The LabelForm object contains information for labels manually drawn on the chart. To create a LabelForm object, call the LabelForm constructor:

```
var form = new LabelForm();
```

The label border color and style are set by the inherited *GraphForm.setColor(value)* and *GraphForm.setLine(value)* properties.

Full Name

```
inetsoft.graph.guide.form.LabelForm
```

LabelForm: setAlignmentX, setAlignmentY, setCollisionModifier, setInsets, setLabel, setPoint, setTextSpec, setTuple,setValues

Inherits from

GraphForm: setColor, setFill, setInsidePlot, setLine, setMeasure, setZIndex

See Also

DefaultTextFrame, to automatically use data values as element labels.

LabelForm.SETALIGNMENTX(VALUE)

Specifies the horizontal alignment of the label with respect to the specified X location. Alignment is also applied to the label text.

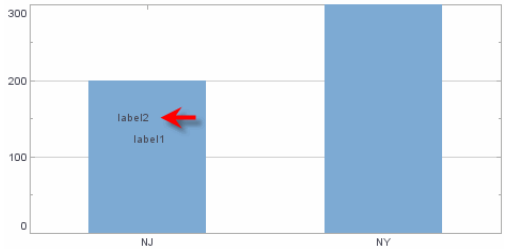
Values

```
Chart.LEFT_ALIGNMENT
Chart.CENTER_ALIGNMENT
Chart.RIGHT_ALIGNMENT
```

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State","Quantity"], ["NJ",200], ["NY",300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var form1 = new LabelForm();
var form2 = new LabelForm();
form1.setLabel("label1");
form1.setPoint(new java.awt.Point(120, 100));
form2.setLabel("label2");
form2.setPoint(new java.awt.Point(120, 120));
form2.setAlignmentX(Chart.CENTER_ALIGNMENT);
graph.addForm(form1);
graph.addForm(form2);
graph.addElement(elem);
```



Inherits from

GraphForm: setColor, setFill, setInsidePlot, setLine, setMeasure, setZIndex

See Also

DataSet, to use a Data Block or query as the *dataset* source.

LabelForm.SETALIGNMENTY(VALUE)

Specifies the vertical alignment of the label with respect to the specified Y location.

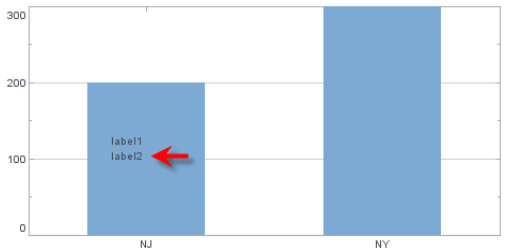
Values

```
Chart.TOP_ALIGNMENT
Chart.MIDDLE_ALIGNMENT
Chart.BOTTOM_ALIGNMENT
```

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State","Quantity"], ["NJ",200], ["NY",300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var form1 = new LabelForm();
var form2 = new LabelForm();
form1.setLabel("label1");
form1.setPoint(new java.awt.Point(100, 100));
form2.setLabel("label2");
form2.setPoint(new java.awt.Point(100, 100));
form2.setAlignmentY(Chart.TOP_ALIGNMENT);
graph.addForm(form1);
graph.addForm(form2);
graph.addElement(elem);
```



Inherits from

GraphForm: setColor, setFill, setInsidePlot, setLine, setMeasure, setZIndex

See Also

DataSet, to use a Data Block or query as the *dataset* source.

LabelForm.SETCOLLISIONMODIFIER(VALUE)

Specifies how collisions (labels occupying the same location) should be handled.

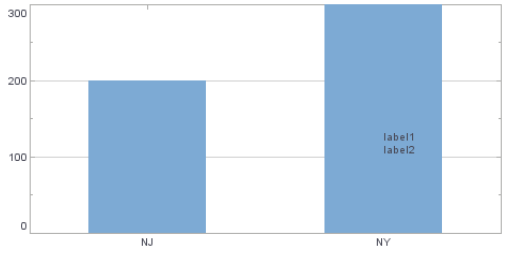
Values

```
VLabel.MOVE_NONE       // no adjustment
VLabel.MOVE_FREE       // move label any direction
VLabel.MOVE_RIGHT      // move label right
VLabel.MOVE_UP         // move label up
```

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State","Quantity"], ["NJ",200], ["NY",300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var form1 = new LabelForm();
var form2 = new LabelForm();
form1.setLabel("label1");
form1.setValues(['NY', 100]);
form2.setLabel("label2");
form2.setValues(['NY', 100]);
form1.setCollisionModifier(VLabel.MOVE_FREE);
graph.addForm(form1);
graph.addForm(form2);
graph.addElement(elem);
```



Inherits from

GraphForm: setColor, setFill, setInsidePlot, setLine, setMeasure, setZIndex

See Also

DataSet, to use a Data Block or query as the *dataset* source.

LabelForm.SETINSETS(VALUE)

Specifies the padding in pixels surrounding the label text.

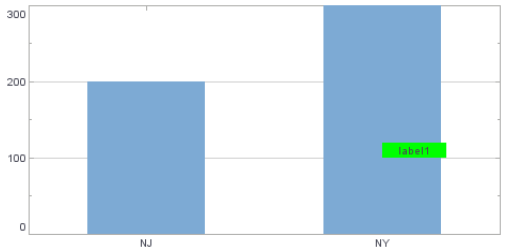
Type

`value` a `java.awt.Insets` object

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State","Quantity"], ["NJ",200], ["NY",300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var form = new LabelForm();
form.setLabel("label1");
form.setValues(['NY', 100]);
var spec = new TextSpec();
spec.setBackground(java.awt.Color(0x00ff00));
form.setTextSpec(spec);
form.setInsets(new java.awt.Insets(0,15,0,15));
// Argument order: top,left,bottom,right
graph.addForm(form);
graph.addElement(elem);
```



Inherits from

GraphForm: setColor, setFill, setInsidePlot, setLine, setMeasure, setZIndex

See Also

DataSet, to use a Data Block or query as the *dataset* source.

LabelForm.SETLABEL(VALUE)

Specifies the text of the label. (Use "\n" to insert newline.)

Type

`value` a String containing label text

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ", 200], ["NY", 300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var form = new LabelForm();
form.setLabel("Hello\nWorld");
form.setValues(['NY', 100]);
graph.addForm(form);
graph.addElement(elem);
```

Inherits from

GraphForm: setColor, setFill, setInsidePlot, setLine, setMeasure, setZIndex

See Also

DataSet, to use a Data Block or query as the *dataset* source.
DefaultTextFrame, to automatically use data values as element labels.

LabelForm.SETPOINT(VALUE)

Specifies the pixel location (integer values) or proportional location (fractional values) for the label. (Positive values specify distance from left/bottom. Negative values specify distance from right/top.)

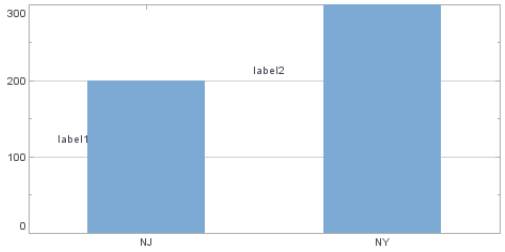
Type

value	a subclass of java.awt.geom.Point2D e.g., java.awt.Point for pixels java.awt.geom.Point2D.Double for proportion

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ", 200], ["NY", 300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var form1 = new LabelForm();
var form2 = new LabelForm();
form1.setLabel("label1");
form2.setLabel("label2");
// Set form1 points in pixels:
form1.setPoint(new java.awt.Point(50, 100));
// Set form2 points proportionally:
form2.setPoint(new java.awt.geom.Point2D.Double(.5,.7));
graph.addForm(form1);
graph.addForm(form2);
graph.addElement(elem);
```



Inherits from
GraphForm: setColor, setFill, setInsidePlot, setLine, setMeasure, setZIndex

See Also
DataSet, to use a Data Block or query as the *dataset* source.

LabelForm.SETTEXTSPEC(SPEC)

Specifies the label text attributes, such as color, font, format, etc.

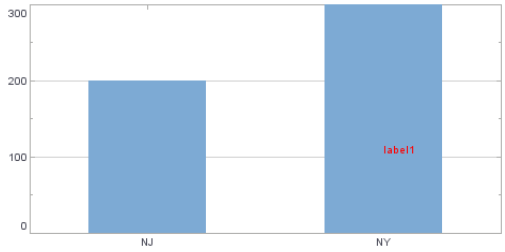
Type

spec a *TextSpec* object

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ", 200], ["NY", 300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var form = new LabelForm();
form.setLabel("label1");
form.setValues(['NY', 100]);
var spec = new TextSpec();
spec.setColor(new java.awt.Color(0xff0000));
form.setTextSpec(spec);
graph.addForm(form);
graph.addElement(elem);
```



Inherits from
GraphForm: setColor, setFill, setInsidePlot, setLine, setMeasure, setZIndex

See Also
DataSet, to use a Data Block or query as the *dataset* source.

LabelForm.SETTUPLE(VALUE)

Specifies the point in logical space for the label text. The coordinates of the point are relative to the prevailing axis scaling.

Type

value	An [X,Y] pair

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ", 200], ["NY", 300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var form = new LabelForm();
form.setLabel("label1");
form.setTuple([0, 100]);
graph.addForm(form);
graph.addElement(elem);
```

Inherits from
GraphForm: setColor, setFill, setInsidePlot, setLine, setMeasure, setZIndex

See Also
DataSet, to use a Data Block or query as the *dataset* source.

LabelForm.SETVALUES(VALUE)

Specifies the location of the label text using coordinate values, numeric or categorical. The coordinates of the point are relative to the prevailing axis scaling. So, for a categorical X-axis (e.g., 'NJ', 'NY', 'PA', etc.), the X-value of `setValues` should specify a categorical value (e.g., 'NJ').

Type

value	an [X,Y] pair

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"],["NJ", 200],["NY", 300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var form = new LabelForm();
form.setLabel("label1");
form.setValues(['NY', 100]);
```

```
graph.addForm(form);
graph.addElement(elem);
```

Inherits from

GraphForm: setColor, setFill, setInsidePlot, setLine, setMeasure, setZIndex

See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.8.5 TagForm

The TagForm object contains information for a tag form. A tag is a label that is associated with a particular object and is automatically positioned to avoid other objects.

To create a TagForm object, call the TagForm constructor:

```
var tag = new TagForm();
```

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset =
[["State","Quantity"],["NJ",200],["NY",300],["NY",305]];
graph = new EGraph();
var elem = new PointElement("State", "Quantity");
var form1 = new TagForm();
var form2 = new TagForm();
form1.setLabel("label1");
form1.setValues(['NY', 300]);
form2.setLabel("label2");
form2.setValues(['NY', 305]);
graph.addForm(form1);
graph.addForm(form2);
graph.addElement(elem);
```

Full Name

```
inetsoft.graph.guide.form.TagForm
```

Inherits from

LabelForm: setAlignmentX, setAlignmentY, setCollisionModifier, setInsets, setLabel, setPoint, setTextSpec, setTuple,setValues
GraphForm: setColor, setFill, setInsidePlot, setLine, setMeasure, setZIndex

See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.8.6 ShapeForm

The ShapeForm object contains information for shapes manually drawn on the chart.

Full Name

`inetsoft.graph.guide.form.ShapeForm`

ShapeForm: setAlignmentX, setAlignmentY, setPosition, setRotation, setShape, setSize, setTuple, setValues

Inherits from

GraphForm: setColor, setFill, setInsidePlot, setLine, setMeasure, setZIndex

ShapeForm.SETALIGNMENTX(VALUE)

Specifies the horizontal alignment of the shape with respect to the specified X position.

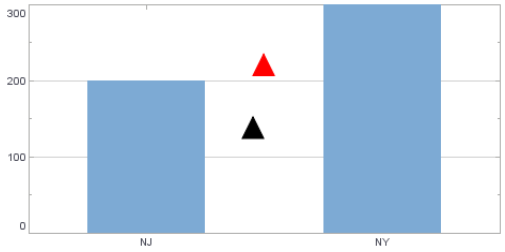
Values

```
Chart.LEFT_ALIGNMENT
Chart.CENTER_ALIGNMENT
Chart.RIGHT_ALIGNMENT
```

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State","Quantity"], ["NJ",200], ["NY",300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var shape1 = new ShapeForm();
var shape2 = new ShapeForm();
shape1.setPoint(new java.awt.geom.Point2D.Double(.5,.75));
shape1.setShape(GShape.FILLED_TRIANGLE);
shape1.setColor(java.awt.Color(0xff0000));
shape1.setSize(new java.awt.Dimension(20,20));
shape1.setAlignmentX(Chart.LEFT_ALIGNMENT);
shape2.setPoint(new java.awt.geom.Point2D.Double(.5,.5));
shape2.setShape(GShape.FILLED_TRIANGLE);
shape2.setColor(java.awt.Color(0x000000));
shape2.setSize(new java.awt.Dimension(20,20));
shape2.setAlignmentX(Chart.CENTER_ALIGNMENT);
graph.addForm(shape1);
graph.addForm(shape2);
graph.addElement(elem);
```



Inherits from

GraphForm: setColor, setFill, setInsidePlot, setLine, setMeasure, setZIndex

See Also

DataSet, to use a Data Block or query as the *dataset* source.

ShapeForm.SETALIGNMENTY(VALUE)

Specifies the vertical alignment of the shape with respect to the specified Y position.

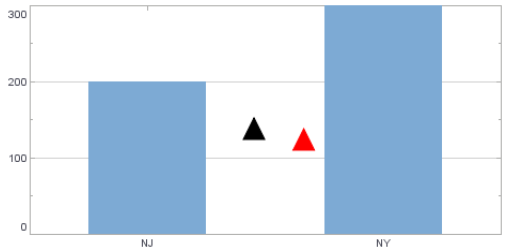
Values

```
Chart.TOP_ALIGNMENT
Chart.MIDDLE_ALIGNMENT
Chart.BOTTOM_ALIGNMENT
```

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State","Quantity"], ["NJ",200], ["NY",300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var shape1 = new ShapeForm();
var shape2 = new ShapeForm();
shape1.setPoint(new java.awt.geom.Point2D.Double(.6,.5));
shape1.setShape(GShape.FILLED_TRIANGLE);
shape1.setColor(java.awt.Color(0xff0000));
shape1.setSize(new java.awt.Dimension(20,20));
shape1.setAlignmentY(Chart.TOP_ALIGNMENT);
shape2.setPoint(new java.awt.geom.Point2D.Double(.5,.5));
shape2.setShape(GShape.FILLED_TRIANGLE);
shape2.setColor(java.awt.Color(0x000000));
shape2.setSize(new java.awt.Dimension(20,20));
shape2.setAlignmentY(Chart.MIDDLE_ALIGNMENT);
graph.addForm(shape1);
graph.addForm(shape2);
graph.addElement(elem);
```



Inherits from

GraphForm: setColor, setFill, setInsidePlot, setLine, setMeasure, setZIndex

See Also

DataSet, to use a Data Block or query as the *dataset* source.

ShapeForm.SETPOINT(VALUE)

Specifies the pixel location or proportional location where the shape is placed. (Positive values specify distance from left/bottom. Negative values specify distance from right/top.)

Type

value	a subclass of java.awt.geom.Point2D e.g., java.awt.Point for pixels java.awt.geom.Point2D.Double for proportion

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

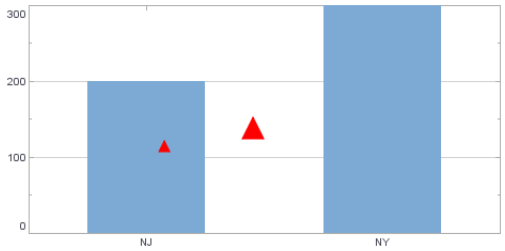
Example (Report or Viewsheet)

```
dataset = [["State","Quantity"], ["NJ",200], ["NY",300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var shape1 = new ShapeForm();
var shape2 = new ShapeForm();

// Set shape1 points in pixels:
shape1.setPoint(new java.awt.Point(150, 100));
shape1.setShape(GShape.FILLED_TRIANGLE);
shape1.setColor(java.awt.Color(0xff0000));
shape1.setSize(new java.awt.Dimension(10,10));

// Set shape2 points proportionally:
shape2.setPoint(new java.awt.geom.Point2D.Double(.5,.5));
shape2.setShape(GShape.FILLED_TRIANGLE);
shape2.setColor(java.awt.Color(0xff0000));
shape2.setSize(new java.awt.Dimension(20,20));

graph.addForm(shape1);
graph.addForm(shape2);
graph.addElement(elem);
```



Inherits from

GraphForm: setColor, setFill, setInsidePlot, setLine, setMeasure, setZIndex

See Also

DataSet, to use a Data Block or query as the *dataset* source.

ShapeForm.SETROTATION(VALUE)

Specifies the shape rotation in degrees.

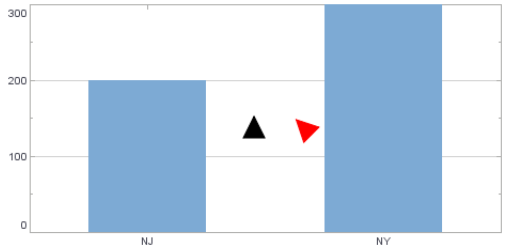
Type

Number

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State","Quantity"], ["NJ",200], ["NY",300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var shape1 = new ShapeForm();
var shape2 = new ShapeForm();
shape1.setPoint(new java.awt.geom.Point2D.Double(.6,.5));
shape1.setShape(GShape.FILLED_TRIANGLE);
shape1.setColor(java.awt.Color(0xff0000));
shape1.setSize(new java.awt.Dimension(20,20));
shape1.setRotation(45);
shape2.setPoint(new java.awt.geom.Point2D.Double(.5,.5));
shape2.setShape(GShape.FILLED_TRIANGLE);
shape2.setColor(java.awt.Color(0x000000));
shape2.setSize(new java.awt.Dimension(20,20));
graph.addForm(shape1);
graph.addForm(shape2);
graph.addElement(elem);
```



Inherits from

GraphForm: setColor, setFill, setInsidePlot, setLine, setMeasure, setZIndex

See Also

DataSet, to use a Data Block or query as the *dataset* source.

ShapeForm.SETSHAPE(SHAPE)

Specifies the type of shape as a *GShape* or *SVGShape*.

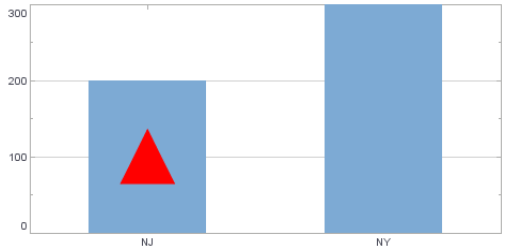
Type

GShape
SVGShape
GShape.ImageShape

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

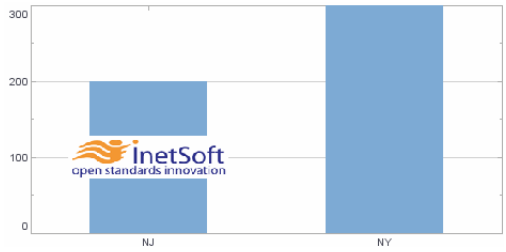
Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ", 200], ["NY", 300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var form = new ShapeForm();
form.setValues(['NJ',100]);
form.setShape(GShape.FILLED_TRIANGLE);
form.setColor(java.awt.Color(0xff0000));
form.setSize(new java.awt.Dimension(50,50));
graph.addForm(form);
graph.addElement(elem);
```



Example (Report or Viewsheet)

```
dataset = [["State","Quantity"], ["NJ",200], ["NY",300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var logo = getImage("http://www.inetsoft.com/images/home/
logo.gif");
var shape = new GShape.ImageShape();
shape.setImage(logo);
var form = new ShapeForm();
form.setValues(['NJ',100]);
form.setShape(shape);
form.setSize(new java.awt.Dimension(150,40));
graph.addForm(form);
graph.addElement(elem);
```



Inherits from
GraphForm: setColor, setFill, setInsidePlot, setLine, setMeasure, setZIndex

See Also
DataSet, to use a Data Block or query as the *dataset* source.

ShapeForm.SETSIZE(VALUE)

Specifies the size of the shape in pixels.

Type
value a `java.awt.Dimension` object

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
var arr = [["State", "Quantity"], ["NJ", 200], ["NY", 300]];
var data = new DataSet(arr);
var graph = new EGraph(data);
var elem = new IntervalElement("State", "Quantity");
var form = new ShapeForm();
form.setValues(['NJ',100]);
form.setShape(GShape.FILLED_TRIANGLE);
form.setColor(java.awt.Color(0xff0000));
form.setSize(new java.awt.Dimension(50,50));
graph.addForm(form);
graph.addElement(elem);
```

Inherits from
GraphForm: setColor, setFill, setInsidePlot, setLine, setMeasure, setZIndex

See Also
DataSet, to use a Data Block or query as the *dataset* source.

ShapeForm.SETTUPLE(VALUE)

Specifies the location of the shape in logical space. The coordinates of the location are relative to the prevailing axis scaling.

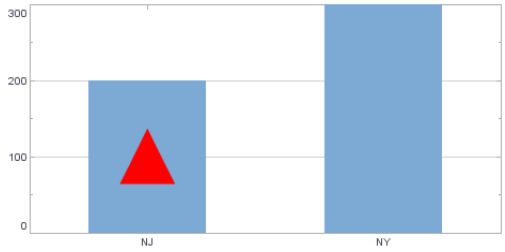
Type

value An [X,Y] pair

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ", 200], ["NY", 300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var form = new ShapeForm();
form.setTuple([.5,100]);
form.setShape(GShape.FILLED_TRIANGLE);
form.setColor(java.awt.Color(0xff0000));
form.setSize(new java.awt.Dimension(50,50));
graph.addForm(form);
graph.addElement(elem);
```



Inherits from

GraphForm: setColor, setFill, setInsidePlot, setLine, setMeasure, setZIndex

See Also

DataSet, to use a Data Block or query as the *dataset* source.

ShapeForm.SETVALUES(VALUE)

Specifies the location of the shape in logical space. The coordinates of the location are relative to prevailing axis scaling prior to transformation. So, for a categorical X-axis (e.g., 'NJ', 'NY', 'PA', etc.), the X-value of the location should specify a categorical value (e.g., 'NJ').

Type

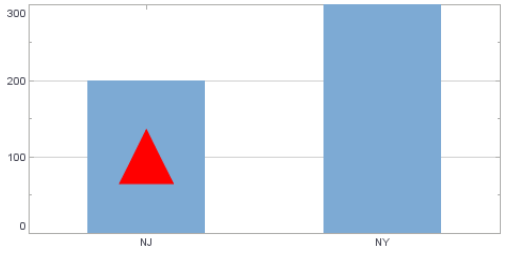
value an [X,Y] pair

Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ", 200], ["NY", 300]];
graph = new EGraph();
```

```
var elem = new IntervalElement("State", "Quantity");
var form = new ShapeForm();
form.setValues(['NJ',100]);
form.setShape(GShape.FILLED_TRIANGLE);
form.setColor(java.awt.Color(0xff0000));
form.setSize(new java.awt.Dimension(50,50));
graph.addForm(form);
graph.addElement(elem);
```



Inherits from

GraphForm: setColor, setFill, setInsidePlot, setLine, setMeasure, setZIndex

See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.9 Utility Objects

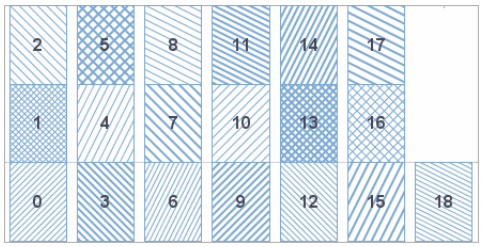
This section describes objects which provide useful constants.

CR.9.1 GTexture

The GTexture object contains a set of patterns. Create a GTexture object by referring to a pattern number, 1–19.

```
var texture = GTexture.PATTERN_5;
```

The image below presents the available textures and their corresponding numbers.



See Also

StaticTextureFrame, for information on using static textures.

CR.9.2 GLine

The GLine object provides the following predefined line style constants:

```
GLine.THIN_LINE
GLine.DOT_LINE
GLine.DASH_LINE
GLine.MEDIUM_DASH
GLine.LARGE_DASH
```

You can also create a GLine object by passing a `Chart` constant (see *Line Styles*) to the object constructor, as follows:

```
var line = new GLine(Chart.DOT_LINE);
```

To create a user-defined style, pass a dash size and width (both type doubles) to the object constructor, as follows.

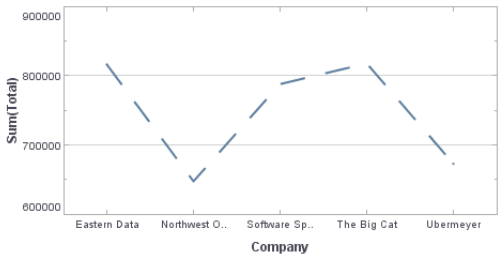
```
var line = new GLine(dashsize, width);
```

```
Note: Report script that modifies 'bindingInfo' should be placed in
the onLoad handler.
```

Example (Report)

Bind a line-type chart to the sample 'All Sales' query, with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the *onLoad Handler*.

```
Chart1.bindingInfo.setShapeField("Company",Chart.NUMBER)
Chart1.bindingInfo.lineFrame = new StaticLineFrame;
Chart1.bindingInfo.lineFrame.line = GLine(20,10);
```



See Also

Line Styles, for the built-in styles available as *Chart Constants* constants.
StaticLineFrame, for information on using GLines.

CR.9.3 GShape

The GShape object contains a set of shapes. For element properties requiring a GShape object, the shape can be specified as shown below.

```
GShape.ImageShape          (user-defined image)
GShape.ARROW
GShape.ARROWBAR
GShape.CIRCLE
GShape.CROSS
GShape.DIAMOND
GShape.FILLED_ARROW
GShape.FILLED_ARROWBAR
```

```
GShape.FILLED_CIRCLE
GShape.FILLED_DIAMOND
GShape.FILLED_SQUARE
GShape.FILLED_TRIANGLE
GShape.HYPHEN
GShape.LINE
GShape.LSHAPE
GShape.SQUARE
GShape.STAR
GShape.STICK
GShape.TRIANGLE
GShape.VSHAPE
GShape.XSHAPE
GShape.NIL (no shape)
```

See Also

SVGShape, for information on using built-in and custom SVG images.
ShapeForm, for information on using GShapes as decorative shapes.
StaticShapeFrame, for information on using GShapes as static shapes.

CR.9.4 GShape.ImageShape

The GShape.ImageShape object contains a custom image to be used as a fill pattern with *StaticShapeFrame* and *CategoricalShapeFrame* objects. To create an ImageShape object, call the object constructor:

```
var shape = new GShape.ImageShape("http://.../image.gif");
```

You can provide the image location as input to the constructor, e.g.,

```
var shape = new GShape.ImageShape("http://.../image.gif");
```

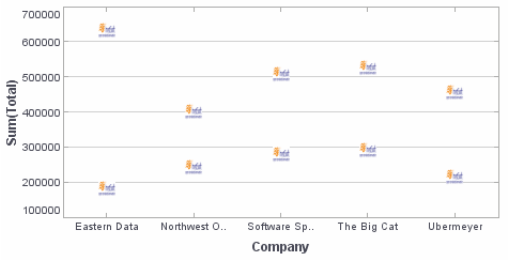
or specify this later using the *GShape.ImageShape.image* property.

```
Note: Report script that modifies 'bindingInfo' should be placed in
the onLoad handler.
```

Example (Report)

Bind a *point-type* chart to the sample ‘All Sales’ query, with ‘Company’ (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

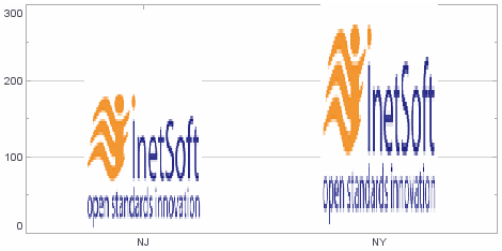
```
Chart1.bindingInfo.setShapeField('Employee',Chart.STRING)
var logo = getImage("http://www.inetsoft.com/images/home/
logo.gif");
var shape = new GShape.ImageShape();
shape.image = logo;
var frame = new StaticShapeFrame(shape);
Chart1.bindingInfo.shapeFrame = frame;
```



Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.

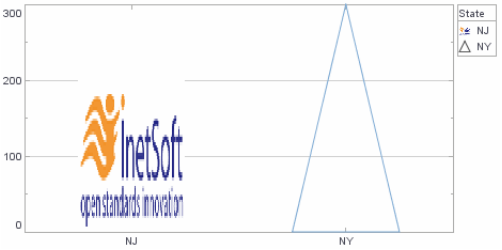
Example (Report or Viewsheet)

```
dataset = [["State","Quantity"], ["NJ",200], ["NY",300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var logo = getImage("http://www.inetsoft.com/images/home/
logo.gif");
var shape = new GShape.ImageShape();
shape.image = logo;
var frame = new StaticShapeFrame(shape)
elem.setShapeFrame(frame);
graph.addElement(elem);
```



Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ", 200], ["NY", 300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var logo = getImage("http://www.inetsoft.com/images/home/
logo.gif");
var shape = new GShape.ImageShape(logo);
var frame = new CategoricalShapeFrame("State");
frame.setShape("NJ", shape);
elem.setShapeFrame(frame);
graph.addElement(elem);
```



Full Name

`inetsoft.graph.aesthetic.GShape.ImageShape`

GShape.ImageShape: setShape, setTile

See Also

DataSet, to use a Data Block or query as the *dataset* source.
StaticShapeFrame, for information on using custom images.

GShape.ImageShape.IMAGE

Specifies the image to use as the fill.

Type

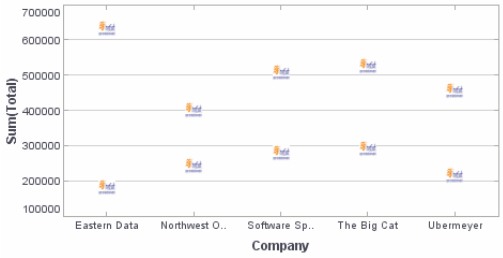
`Image object, see getImage()`

`Note: Report script that modifies 'bindingInfo' should be placed in the onLoad handler.`

Example (Report)

Bind a *point-type* chart to the sample 'All Sales' query, with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

```
Chart1.bindingInfo.setShapeField("Employee",Chart.STRING);
var logo = getImage("http://www.inetsoft.com/images/home/
logo.gif");
var shape = new GShape.ImageShape();
shape.image = logo;
var frame = new StaticShapeFrame(shape);
Chart1.bindingInfo.shapeFrame = frame;
```

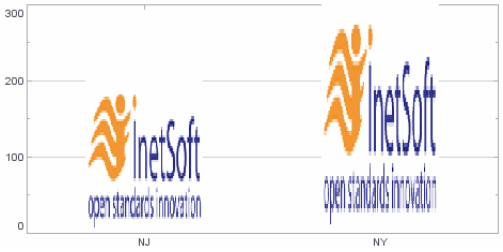


`Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and *Adding Component Script* in *Dashboard Scripting* for more information.`

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ", 200], ["NY", 300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var logo = getImage("http://www.inetsoft.com/images/home/
logo.gif");
var shape = new GShape.ImageShape();
shape.setImage(logo);
```

```
var frame = new StaticShapeFrame(shape)
elem.setShapeFrame(frame);
graph.addElement(elem);
```



See Also

DataSet, to use a Data Block or query as the *dataset* source.

GShape.ImageShape.TILE

Specifies whether the image should be stretched to fit the fill area or tiled at the original size.

Type

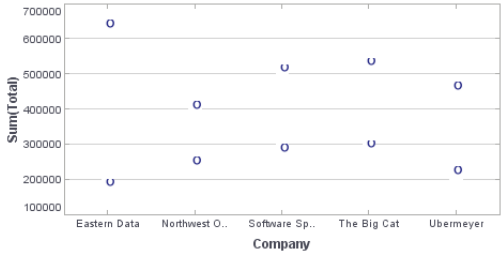
boolean	true: keep original size, and tile to fit false: stretch to fit (default)

Note: Report script that modifies 'bindingInfo' should be placed in the onLoad handler.

Example (Report)

Bind a *point-type* chart to the sample 'All Sales' query, with 'Company' (top 5) on the X-axis, and Sum(Total) on the Y-axis. Add the following script in the onLoad handler.

```
Chart1.bindingInfo.setShapeField("Employee",Chart.STRING);
var logo = getImage("http://www.inetsoft.com/images/home/
logo.gif");
var shape = new GShape.ImageShape();
shape.image = logo;
shape.tile = true;
var frame = new StaticShapeFrame(shape);
Chart1.bindingInfo.shapeFrame = frame;
```

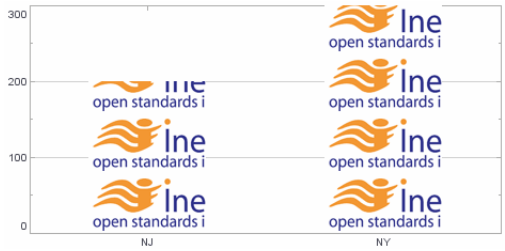


Note: Script that modifies 'graph' should be placed at the element level. See *Adding Element-Level Script* in *Report Scripting* and

Adding Component Script in *Dashboard Scripting* for more information.

Example (Report or Viewsheet)

```
dataset = [["State", "Quantity"], ["NJ", 200], ["NY", 300]];
graph = new EGraph();
var elem = new IntervalElement("State", "Quantity");
var logo = getImage("http://www.inetsoft.com/images/home/
logo.gif");
var shape = new GShape.ImageShape();
shape.setImage(logo);
shape.setTile(true);
var frame = new StaticShapeFrame(shape)
elem.setShapeFrame(frame);
graph.addElement(elem);
```



See Also

DataSet, to use a Data Block or query as the *dataset* source.

CR.9.5 SVGShape

The SVGShape object contains a set of shapes for use with *StaticShapeFrame* and *ShapeForm*. For element properties requiring a SVGShape object, the shape can be specified as shown below.

```
SVGShape.CHECK
SVGShape.DOWN_ARROW
SVGShape.FACE_BLANK
SVGShape.FACE_HAPPY
SVGShape.FACE_OK
SVGShape.FACE_SAD
SVGShape.FACE_SMILE
SVGShape.FEMALE
SVGShape.LEFT_ARROW
SVGShape.MALE
SVGShape.MINUS
SVGShape.PLUS
SVGShape.RIGHT_ARROW
SVGShape.STAR
SVGShape.SUN
SVGShape.UP_ARROW
SVGShape.WARNING
SVGShape.X
SVGShape.NIL (no shape)
```

You can also create a custom SVGShape from an SVG image stored in the local file system or on a server.

Example 1 (Built-in SVG Shape)

```
dataset = [["State","Quantity"], ["NJ",200], ["NY",300]];
graph = new EGraph();
var elem = new PointElement("State","Quantity");
var shapeFrame = new StaticShapeFrame(SVGShape.FACE_HAPPY);
var sizeFrame = new StaticSizeFrame(10);
elem.setSizeFrame(sizeFrame);
elem.setShapeFrame(shapeFrame);
graph.addElement(elem);
```



Example 2 (Local SVG Shape)

```
dataset = [["State","Quantity"], ["NJ",200], ["NY",300]];
graph = new EGraph();
var elem = new PointElement("State","Quantity");
var svg = new SVGShape("file:\C:/HappyFaceSVG.svg");
var shapeFrame = new StaticShapeFrame(svg);
var sizeFrame = new StaticSizeFrame(10);
elem.setSizeFrame(sizeFrame);
elem.setShapeFrame(shapeFrame);
graph.addElement(elem);
```

Example 3(Remote SVG Shape)

```
dataset = [["State","Quantity"], ["NJ",200], ["NY",300]];
graph = new EGraph();
var elem = new PointElement("State","Quantity");
var svg = new SVGShape("http://www.w3.org/Icons/SVG/svg-
logo.svg");
var shapeFrame = new StaticShapeFrame(svg);
var sizeFrame = new StaticSizeFrame(10);
elem.setSizeFrame(sizeFrame);
elem.setShapeFrame(shapeFrame);
graph.addElement(elem);
```

See Also

GShape, for information on using built-in bitmap shapes.
GShape.ImageShape, for information on using custom bitmap shapes.
ShapeForm, for information on using GShapes as decorative shapes.
StaticShapeFrame, for information on using GShapes as static shapes.

CR.10 Z-Index Defaults

The *GraphForm.setZIndex(value)* property of a form object determines whether the object lies above or below other chart objects. When an object has a higher zIndex than a second object, the first object will partially or fully occlude the second object

The table below provides the default zIndex settings for common chart objects.

OBJECT	DEFAULT ZINDEX
Coordinate border	20
Grid line	30
Axis	40
Axis border	50
Visual object	60
Grid line on top of Object	70
Form object	80
Facet gridline	90
Text	100

CR.11 Chart Constants

The Chart object is a static object that contains definitions for constants related to charts (line styles, etc.). These constants can be accessed without explicitly creating an instance of a Chart object, as shown below:

```
Chart1.chartStyle['Sum(Total)'] = Chart.CHART_LINE;
```

CHART STYLES

The following *Chart Constants* style constants are available for the *chartStyle* property within report onLoad and Viewsheet script.

CHART STYLE	DESCRIPTION
CHART_AUTO	Automatic style selection
CHART_PARETO	Pareto chart.
CHART_LINE	Line chart
CHART_LINE_STACK	Stacked line chart
CHART_POINT	Point chart
CHART_BAR	2D bar chart
CHART_BAR_STACK	Stacked bar chart
CHART_PIE	2D pie chart
CHART_3D_BAR	3D bar chart
CHART_3D_BAR_STACK	3D stacked bar chart
CHART_3D_PIE	3D pie chart
CHART_STOCK	High-low-closing chart
CHART_AREA	Area chart
CHART_AREA_STACK	Stacked area chart
CHART_RADAR	Radar chart
CHART_FILL_RADAR	Filled radar chart
CHART_CANDLE	Candle chart
CHART_WATERFALL	Waterfall chart
CHART_MAP	Map chart

LINE STYLES

The *Chart Constants* line style constants below can be used in all chart properties requiring line styles, such as axis grid lines, target lines, etc.

LINE STYLE	DESCRIPTION
NONE	No line
ULTRA_THIN_LINE	Very thin line at ¼ of a point width
THIN_THIN_LINE	Very thin line at ½ of a point width
THIN_LINE	Thin line at one point width
MEDIUM_LINE	Medium width line
THICK_LINE	Thick width line
DOT_LINE	Dotted line
DASH_LINE	Dash line
MEDIUM_DASH	Medium size dash line
LARGE_DASH	Large size dash line

See Also
xGridStyle, to set the graph grid lines.

SUMMARIZATION FORMULAS

The *Chart Constants* summarization formulas below are available for chart scripting.

NONE	NTHMOSTFREQUENT
AVERAGE	NTHSMALLEST
CONCAT	POPULATIONSTANDARDDEVIATION
CORRELATION	POPULATIONVARIANCE
COUNT	PRODUCT
COVARIANCE	PTHPERCENTILE
DISTINCTCOUNT	STANDARDDEVIATION
MAX	SUMSQ
MEDIAN	SUMWT
MIN	SUM
MODE	VARIANCE
NTHLARGEST	WEIGHTEDAVERAGE

LEGEND POSITIONS

The following legend positions are available for *EGraph.setLegendLayout(value)*.

LEGEND POSITION	DESCRIPTION
Chart.NONE	No legend
Chart.BOTTOM	Below X-axis title, aligned left
Chart.TOP	Above graph, aligned left

Chart.LEFT	To left of Y-axis title, aligned top
Chart.RIGHT	To right of graph, aligned top (default)
Chart.IN_PLACE	Superimposed on graph

CR.12 Miscellaneous Chart Functions

This section presents special-purpose charting functions.

CREATEBULLETGRAPH(MEASURE, RANGES, TARGET, COLOR, XDIMS, YDIMS, OPTS)

The `createBulletGraph()` function generates a bullet graph to display a specified measure along with a target and multiple value ranges. (All parameters are optional. To omit any parameter setting, pass `null` as the value.)

Parameter

measure	Name of column to provide the measure values
ranges	Array of column names to provide range values, displayed as shaded regions
target	Name of column to provide the target value, displayed as vertical bar
color	A single color or array of colors to be used for bullets. See below.
xdims	Array of column names to use as dimensions on X-axis
ydims	Array of column names to use as dimensions on Y-axis
opts	String containing additional options: 'vertical=false': force horizontal bars 'ylabel=false': hide y-dimension labels 'xlabel=false': hide x-dimension labels 'vlabel=false': hide the measure value labels

The `color` parameter can be either a scalar hex color value (e.g., `0x0000FF`.), or an array of colors. The organization for the color array is as follows:

```
[measure, target, range 1, range 2, range 3, ...]

e.g., [0x0000FF, 0xFF0000, 0xDD0000, 0xBB0000, 0x990000]
```

Example

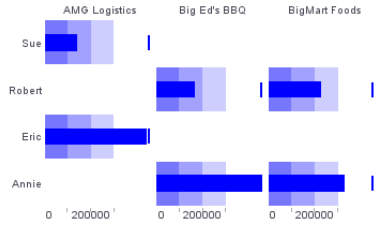
Consider the Worksheet Data Block called, ‘Query1’, shown below. (This Data Block is a mirror of the ‘All Sales’ query, with grouping on the ‘Employee’ and ‘Company’ fields, aggregation on the ‘Total’ field, and expression columns for ‘Target’, ‘range1’, ‘range2’, and ‘range3’.)

Query1

Employee	Company	Total	Target	range1	range2	range3
Annie	Big Ed's BBQ	454367.00	450000	100000	100000	100000
Annie	BigMart Foods	327642.00	450000	100000	100000	100000
Eric	AMG Logistics	442419.00	450000	100000	100000	100000
Robert	Big Ed's BBQ	168172.00	450000	100000	100000	100000
Robert	BigMart Foods	228027.00	450000	100000	100000	100000
Sue	AMG Logistics	143479.00	450000	100000	100000	100000

Create a Viewsheet based on this Data Worksheet, add a Chart component, and insert the following Chart script:

```
dataset = viewsheet['Query1']
graph = createBulletGraph('Total',
['range1','range2','range3'],'Target',0x0000FF,
['Company'],['Employee'],'vertical=false');
```



See Also

DataSet, for information on specifying the chart data in script.

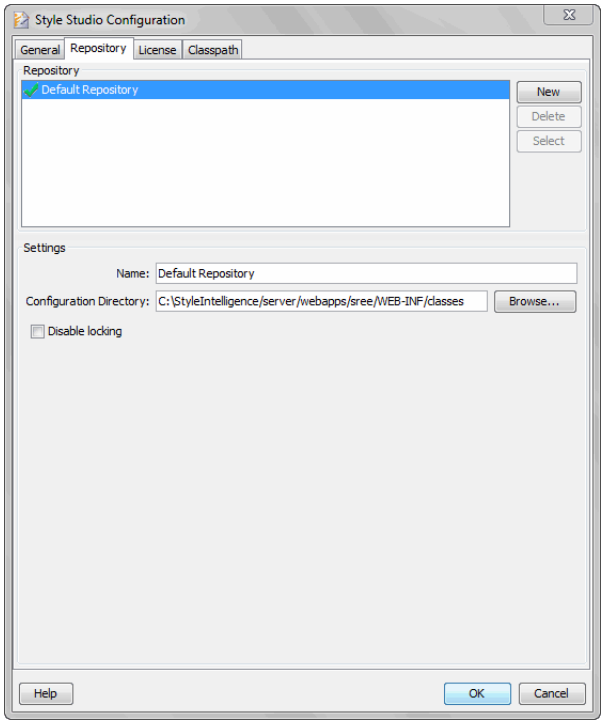
APPENDIX ST: Using Style Studio

This appendix provides more advanced information on Style Studio configuration and deployment issues.

ST.1 Configuring Style Studio

Before you begin working with Style Studio, there are a few settings that you should customize. The most important of these is the Style Studio *working repository*, the location from which Style Studio accesses queries, data models, and other assets.

To change Style Studio configuration settings, select 'Configure' from the 'File' menu. This opens the 'Style Studio Configuration' dialog box.



The 'Style Studio Configuration' dialog box has four tabs: **General**, **Repository**, **License**, and **Classpath**. These tabs allow you to configure multiple repositories, adjust the Style Studio classpath, and tailor other settings. The tabs are discussed in the following sections.

Configuration settings are saved in the *.stylereport* and *.stylereport_repos* files. These files are located in a version-specific directory (e.g., "*11.5*") within the *.stylereport.d* folder in the home (user) directory on the machine where Style Studio is installed. Some of these settings can be viewed in the **System Info** tab of the 'About Style Studio' dialog box, which you can access from the main 'Help' menu.

ST.1.1 Configuring a Repository

A *repository* is a location from which Style Studio accesses assets. You can configure multiple repositories, but only one repository at a time can be the *working repository* (see below). A repository contains the following key configuration files:

These files are accessed both by Style Studio and any Style Scope server that uses the same repository.

- **repository.xml**: The repository listing, containing information about folders displayed under the 'Repository' node of the Style Studio **Asset** panel. The *repository.xml* file is also used to populate the 'Repository' listing for any server that uses this repository.
- **asset.dat**: The asset registry, containing global Data Worksheets (as well as Viewsheets created on the server).

The *asset.dat* and *stylereport.srl* files can be exploded into their individual components, if needed.

- **stylereport.srl**: The library, containing reusable dashboard components (table styles, scripts, etc.).
- **query.xml**: The query registry, containing global query definitions that can be shared among different Data Worksheets and dashboards.
- **datasource.xml**: The data source registry, containing global definitions for database connections, data models, and virtual private models (VPMs) that can be shared among different Data Worksheets and dashboards.

```
Note: A "remote repository" is any repository other than the
current working repository. The remote repository need not be
located on a remote machine.
```

You can configure multiple repositories from within Style Studio (see *Creating a New Repository*), but only one repository at a time can be the current working repository. The *working repository* is the location in which Style Studio is currently set to access the resources described above. Any repository *other* than the working repository is a *remote repository.*

The following sections explain how to configure a repository, and how to select the working repository. To save an asset to a remote repository, see *Deploying an Asset*.

ST.1.2 Creating a New Repository

The *working repository* is the location from which Style Studio *currently* accesses configuration files. A *remote repository* is any repository *other than* the working repository.

A remote repository can have type 'Local'. "Remote" means "not the working repository".

A repository (*working* or *remote*) can be one of three types:

- **Local**: Repository files are stored on a local drive or shared network drive.

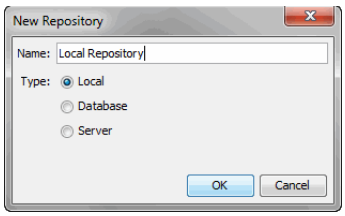
- **Database**: Repository files are stored in a table within a supported database system (Derby, Oracle, SQL Server, DB2, PostgreSQL, MySQL).
- **Server**: Repository files are managed by a server and accessed over the internet.

The following sections explain how to create each type of repository.

ST.1.3 Creating a Local Repository

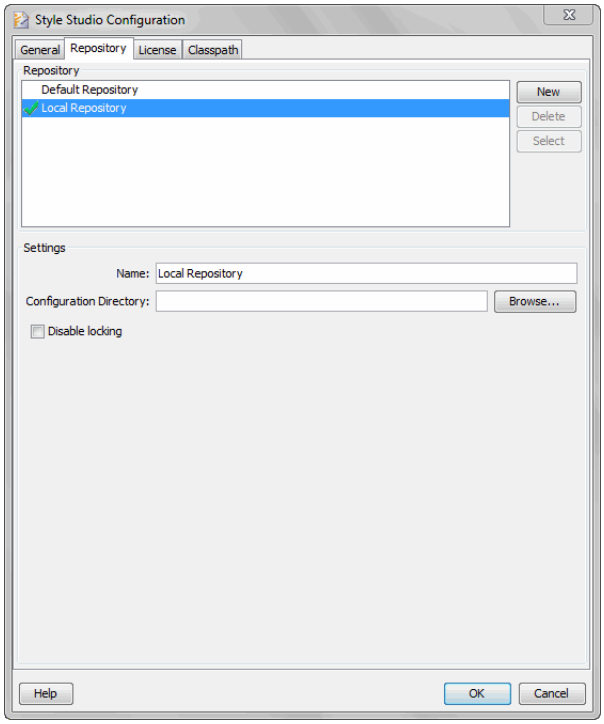
A *local repository* stores the configuration files on a local drive or shared network drive. To create a local repository, follow the steps below:

1. From the Style Studio ‘File’ menu, select ‘Configure’. This opens the ‘Style Studio Configuration’ dialog box.
2. Select the **Repository** tab.
3. On the **Repository** tab, press the ‘New’ button. This opens the ‘New Repository’ dialog box.
4. In the ‘Name’ field, enter the label under which the repository should be listed.
5. Select the ‘Local’ option from the ‘Type’ listing, and press ‘OK’.



A green check-mark indicates the working repository.

This adds a new local repository to the listing in the **Repository** tab, and automatically sets this new repository to be the working repository. (See *Selecting the Working Repository* for information on manually setting the working repository.)




```
The specified directory cannot be used for any other repository.
```

6. In the ‘Configuration Directory’ field, enter the full path to the desired repository directory (a local or network drive location). If the specified directory is empty, a new set of configuration files is created in that location. (See *Creating a Clean Local Repository*.)

7. (Optional) Select ‘Disable Locking’ to disable Style Studio’s automatic locking mechanism. The locking mechanism prevents multiple developers from simultaneously editing the same assets (e.g., data models, queries, etc.), which might lead to edit conflicts. In general, leave the locking mechanism enabled.

8. Press ‘OK’ to close the dialog box.

This new ‘Local’ repository can be used either as the working repository or as a remote repository.

ST.1.4 Creating a Database Repository

A *database repository* stores reports and configuration files in a database schema. To create a database repository, follow the steps below:

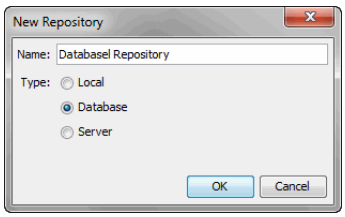
1. From the Style Studio ‘File’ menu, select ‘Configure’. This opens the ‘Style Studio Configuration’ dialog box.

2. Select the **Repository** tab.

3. On the **Repository** tab, press the ‘New’ button. This opens the ‘New Repository’ dialog box.

4. In the ‘Name’ field, enter the label under which the repository should be listed.

5. Select the ‘Database’ option from the ‘Type’ listing, and press ‘OK’.



A green check-mark indicates the working repository.

This adds a new database repository to the listing in the **Repository** tab, and automatically sets this new repository to be the working repository. (See *Selecting the Working Repository* for information on manually setting the repository.)

```
The specified directory cannot be used for any other repository.
```

6. In the ‘Configuration Directory’ field, enter the full path to a directory which will store the database connection information.

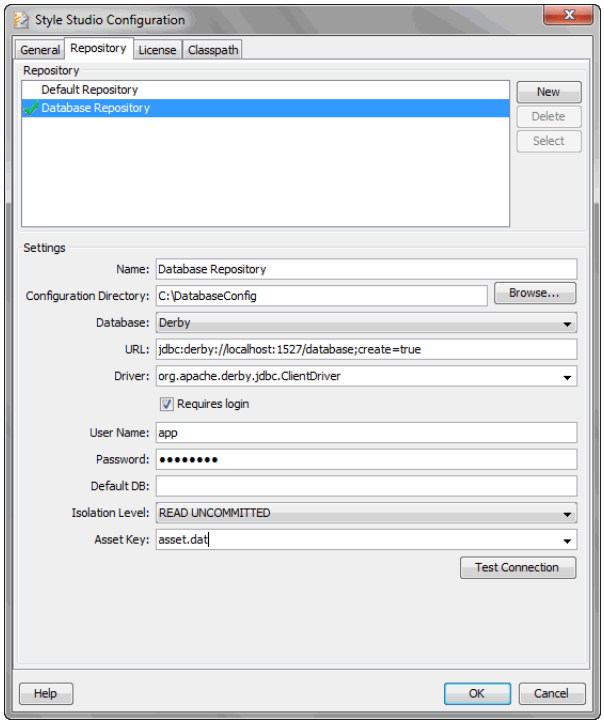
7. In the ‘Database’ menu, select the type of database to use for storage of the repository files. The options are ‘SQLServer’, ‘Oracle’, ‘Derby’, ‘DB2’, PostgreSQL, MySQL. (To use the sample audit database of the default installation, select ‘Derby’.)

Your database administrator can provide the connection URL.

8. In the ‘URL’ field, enter the connection information for the database schema. (To use the sample audit database, enter `jdbc:derby://localhost:1527/database;create=true`. To run the audit database, expand the ‘Style Scope’ program group in the Windows ‘Start’ menu, and select ‘Audit Database’.)

9. In the ‘Driver’ field, enter the appropriate class name for the Java driver. (This field may already be populated based on your selection in the ‘Database’ menu. To use the sample audit database, enter `org.apache.derby.jdbc.ClientDriver`.)

10. Select ‘Requires Login’ if the database requires a user name and password to open a connection. In this case, also enter the desired ‘User Name’ and ‘Password’ credentials. (To use the sample audit database, enter “`app`” for the ‘User Name’ and “`password`” for the ‘Password’.)



11. (Optional) In the ‘Default DB’ field, enter the name of the default database schema to receive the connection.

12. In the ‘Isolation Level’ field, select the desired isolation level.

13. (Optional) If an existing schema contains multiple asset (Worksheet/Viewsheet) packages, specify the desired package in the ‘Asset File Key’ field.

14. Press ‘OK’ to close the dialog box.

This new ‘Database’ repository can be used either as the working repository or as a remote repository.

ST.1.5 Creating a Server Repository

A *server repository* stores reports and configuration files in a server repository. This method uses HTTP for communication, and requires only internet access and a password. The server repository option can support highly distributed development teams working across geographically diverse regions, although performance may be inferior to that of local or database repository options.

To create a server repository, follow the steps below:

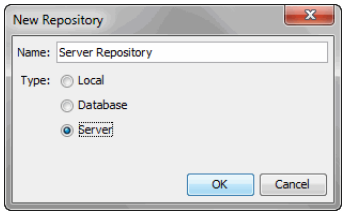
1. From the Style Studio ‘File’ menu, select ‘Configure’. This opens the ‘Style Studio Configuration’ dialog box.

2. Press the **Repository** tab.

3. On the **Repository** tab, press the ‘New’ button. This opens the ‘New Repository’ dialog box.

4. In the ‘Name’ field, enter the label under which the repository should be listed.

5. Select the ‘Server’ option from the ‘Type’ listing, and press ‘OK’.



A green check-mark indicates the working repository.

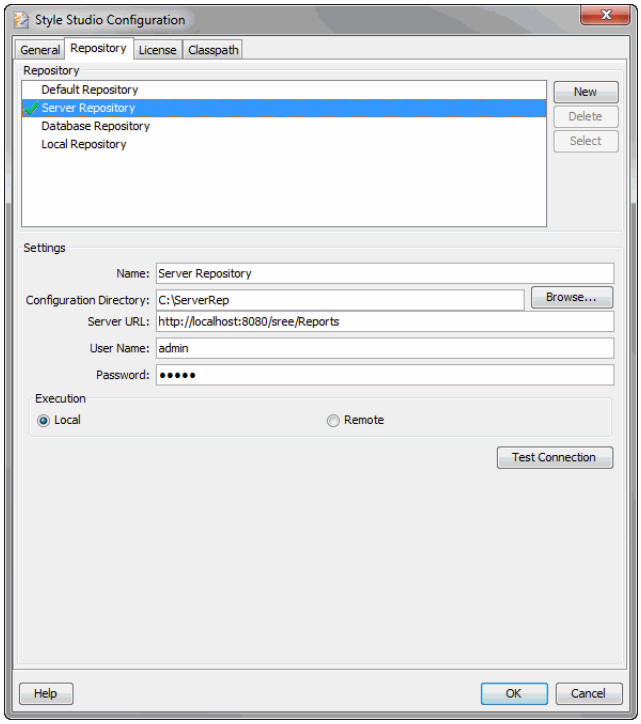
This adds a new server repository to the listing in the **Repository** tab, and automatically sets this new repository to be the working repository. (See *Selecting the Working Repository* for information on manually setting the repository.)

```
The specified directory cannot be used for any other repository.
```

6. In the ‘Configuration Directory’ field, enter the full path to the directory which will store the server connection information.

7. In the ‘Server URL’ menu, enter the full URL of the server to manage the shared repository. (The default URL of the server is `http://hostname:8080/sree/Reports`.) The specified server’s repository will then be used as the Style Studio repository.

8. Enter the ‘User Name’ and ‘Password’ credentials to access the server.



Note: For both 'Local' and 'Remote' options, the repository is managed by the remote server.

9. (Optional) In the 'Execution' panel, select 'Local' if you want Style Studio to perform all query processing on the local machine (default). Select 'Remote' if you want the specified remote server to perform all query processing. This option requires administrative permission.

   Note: If you are using a text or Excel file as a datasource and select the 'Remote' execution option, the datasource file path must be accessible to the server in order to preview queries, etc.

10. Press 'OK' to close the dialog box.

This new 'Server' repository can be used either as the working repository or as a remote repository.

ST.1.6 Selecting the Working Repository

The *working repository* is the repository from which Style Studio is currently accessing assets and configuration settings. To select the working repository, follow the steps below:

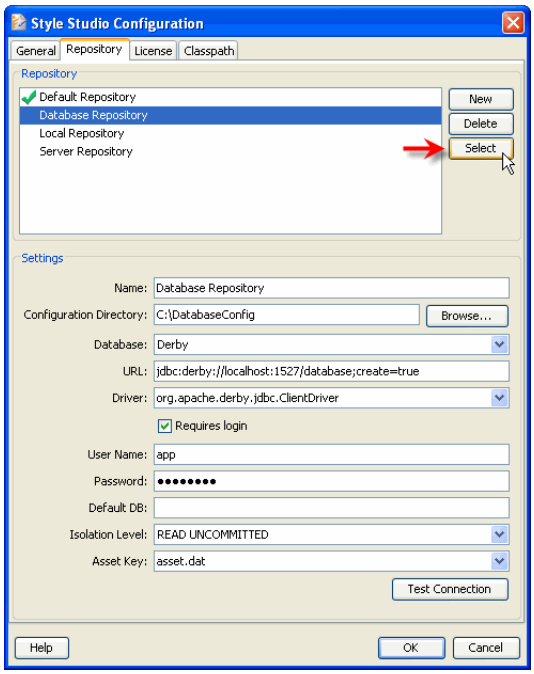
1. From the Style Studio 'File' menu, select 'Configure'. This opens the 'Style Studio Configuration' dialog box.

2. Select the **Repository** tab.

The **Repository** tab lists all of the configured repositories. See *Creating a New Repository* for information on how to add a new repository to this list.

You can also double-click the desired repository.

3. Click the desired repository and press ‘Select’.



The repository listing is stored in the *.stylereport_repos* file. See *Configuring Style Studio* for more details.

Note that a green check mark appears next to the currently selected repository. Style Studio uses this working repository to access the key configuration files described in *Configuring a Repository* (*query.xml*, *repository.xml*, *datasource.xml*, etc.)

4. Press ‘OK’ to close the dialog box.

ST.1.7 Sharing a Repository

To save assets into a repository other than the working repository, see *Deploying an Asset*.

If a team of developers is collaborating on creating assets (queries, models, scripts, etc.) for a particular project, it is best for all developers share the same working repository. This means that each developer should configure and select the same working repository in their own installation of Style Studio. See *Configuring a Repository* for information on how to do this.

By sharing the same working repository, multiple developers can edit the same set of configuration files (*datasource.xml*, *query.xml*, *stylereport.srl*), which simplifies deployment and eliminates synchronization problems.

See *Configuring a Repository* for information on selecting a repository type.

Any type of repository (local, database, or server) can be shared among developers. However, one type of repository may be better suited than another to distributed team development in a particular context. The

following sections explain the advantages and disadvantages of the different repository types for team development.

Sharing a Local Repository

To share a working repository using the 'Local' repository option, mount a network drive and create a directory on this network drive for the repository. Each developer should follow the steps given in *Creating a Local Repository* and specify this network drive directory as their 'Configuration Directory'.

```
Note: For the 'Local' repository approach, set the 'locklevel.user'
property in the 'sree.properties' file to 'false' (default).
```

A useful variation on this approach is to set the 'Configuration Directory' to the SREE Home directory of a development server (e.g., *sree/WEB-INF/classes*). This ensures that modifications you make to data sources, queries, assets, etc., will be available to the server without any special deployment. However, be aware that Style Studio loads the server's *sree.properties* settings with a higher priority than those in the *.stylereport* file in the user home directory, and Style Studio does not write back to *sree.properties*.

It is prudent to implement a backup procedure for network drives.

The network drive approach to a shared repository is very straightforward if your organization already uses network drives. However, this solution may not be feasible if a development team is distributed across more than one office, or if members do not have access to a single network drive. In this case, consider using the 'Database' repository option described below.

Sharing a Database Repository

To share a working repository using the 'Database' repository option, set up a database (SQLServer, Oracle, Derby, DB2, PostgreSQL, MySQL) to store the configuration files. Each developer should follow the steps given in *Creating a Database Repository*, and specify the same database schema information.

```
Note: For 'Database Repository', set the 'locklevel.user' property
in the 'sree.properties' file to 'true'.
```

Databases are usually accessible from multiple locations within an organization, so the database repository approach often readily supports distributed teams. In addition, your organization may already have a backup procedure in place for its databases (which might not be the case for a network drive).

Sharing a Server Repository

To share a working repository using the 'Server' repository option, each developer should follow the steps given in *Creating a Server Repository*, and specify the same 'Server URL' for the servlet.

Because the server repository is accessible to any developer with an internet connection and login credentials, this option can easily support the most distributed of teams. However, performance of the 'Server' repository is generally inferior to that of the 'Local' and 'Database' repositories due to its reliance on HTTP for communication.

Team Development Using Independent Repositories

If your team cannot set up a *shared repository* as recommended in *Sharing a Repository*, it is still possible for multiple developers to create assets using *independent repositories*. However, if developers use independent repositories, they will need to merge the assets that they have individually created.

The easiest way to merge assets (if developers cannot share a common *working repository*) is to set up a common *remote repository* into which developers can deploy assets. See *Configuring a Repository* to set up a remote repository in Style Studio, and see *Deploying an Asset* for instructions on how to save assets into this remote repository. For example, each developer can define a connection from their own Style Studio installation to a common "server-type" remote repository. When individual developers deploy their assets into that common remote repository, the assets will be automatically merged.

If it is not possible for developers to use *either* a common working repository or a common remote repository, they can merge assets from their independent repositories into a single deployment repository by exporting to a JAR file, as described below.

To merge assets from multiple independent repositories into a single deployment repository, follow the steps below:

1. Each developer should deploy the assets they have created into an independent JAR file. (See *Deploying an Asset to a JAR File*.)

2. Move these JAR files to a location that is accessible to the desired deployment repository.

3. Import and merge the assets from the JAR files into the deployment repository:

    a. To do this from within Style Studio, see *Importing Assets from a JAR File*.

    b. To do this from within Enterprise Manager, see *Incrementally Deploying Assets*.

This merges the assets into the deployment repository. To synchronize the independent developer repositories with the common deployment repository, you can use the same approach in reverse: Export a JAR with the desired assets from the deployment repository, and import the JAR contents into the individual developer repositories. (See *Importing Assets from a JAR File*.)

ST.1.8 Creating a Clean Local Repository

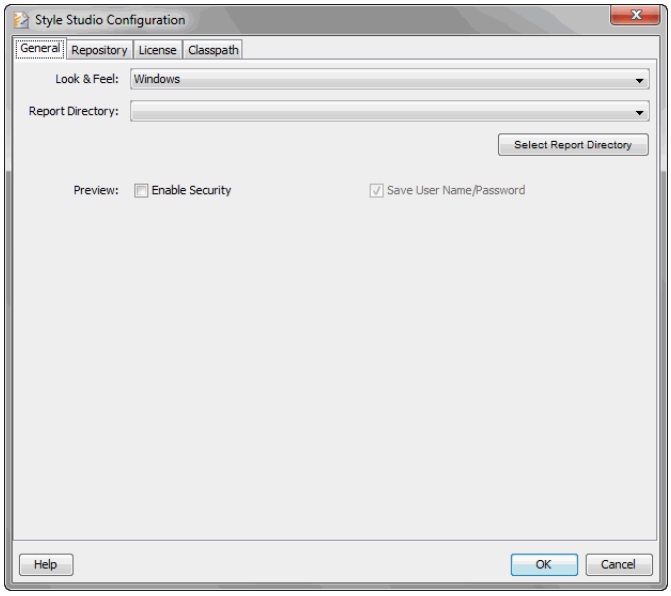
When you first install Style Scope, the repository is set to 'Default Repository', which is a 'Local' repository populated with a number of example data sources, queries, and other assets. The location of this default local repository is *server\webapps\sree\WEB-INF\classes*, which is also the default repository directory for the Style Scope server. (It is useful for Style Studio to share a repository with a development server so that modifications you make to data sources, queries, assets, etc., will be avialable to the server without any special deployment.)

The files in the 'Default Repository' are provided for learning purposes, and the sample queries, data sources, Worksheets, etc., are referenced in the documentation. When you begin to develop your own assets, you may want to create a new repository that is not cluttered by these sample materials.

To create a "clean" repository, simply follow the instructions in *Creating a New Repository*, and set the 'Configuration Directory' to an empty directory. This will automatically create a new set of blank configuration files (*datasource.xml*, *query.xml*, *stylereport.srl*, etc.) in that location.

ST.1.9 Configuring General Style Studio Properties

The **General** tab of the 'Style Studio Configuration' dialog box allows you to specify several properties relating to the appearance and behavior of Style Studio. To open the dialog box, select 'Configure' from the Style Studio 'File' menu, and select the **General** tab.



The following options are available:

- **Look & Feel**: Sets the general appearance of the Style Studio interface.
- **Single Style Studio Instance**: If enabled, allows only a single instance of Style Studio to run on the desktop.
- **Auto-Save**: Automatically saves a copy of the current asset with the frequency specified by the 'Interval' setting. This helps to protect against data loss in case of a machine failure. When you restart Style Studio after such a failure, Style Studio prompts you to restore files from the auto-saved copies.
- **Preview Server Features**: Enables hyperlinks and other server features from with the preview tab.
- **Parameter Name Unique**: Specifies that every occurrence of a parameter name refers to the same parameter.
- **Enable Security**: Instructs Style Studio to apply security policies (including login challenge) during preview. This is useful for testing VPMs or other user-dependent filters. Authentication and authorization settings are drawn from Enterprise Manager. (See *Configuring Security* for details on creating users, etc.)
- **Save User Name/Password**: Caches the most recently entered user name and password to reuse for subsequent previews (without further prompting). This relieves you from re-entering login credentials when you are repeatedly previewing an asset as the same user.

ST.1.10Configuring the Style Studio License

To install a license key, follow the steps below:

1. Select ‘Configure’ from the Style Studio ‘File’ menu. This opens the ‘Style Studio Configuration’ dialog box.

2. Select the **License** tab.

3. Enter the license key, and press ‘OK’.

ST.1.11Configuring the Style Studio Classpath

The classpath is a list of folders and JAR files that the JVM uses to locate Java resources. To specify the classpath, follow the steps below:

1. Select ‘Configure’ from the Style Studio ‘File’ menu. This opens the ‘Style Studio Configuration’ dialog box.

2. Select the **Classpath** tab.

3. Press ‘Add JAR’ to add a JAR file (e.g., a JDBC driver) to the classpath.

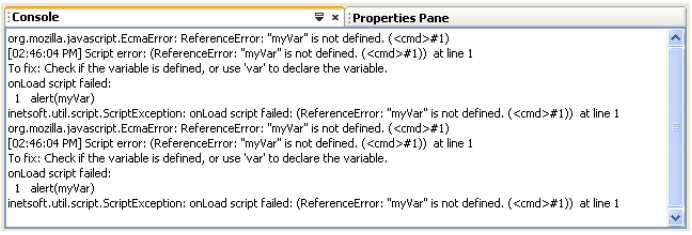
4. Press ‘Add Directory’ to add a file system directory containing resources to the classpath.

5. Restart Style Studio.

The path list that you specify in the **Classpath** tab is stored in the *.stylereport* configuration file. (See *Configuring Style Studio* for more details about the *.stylereport* file.) This same classpath is used whether Style Studio is launched from the Windows ‘Start’ menu or from the command line.

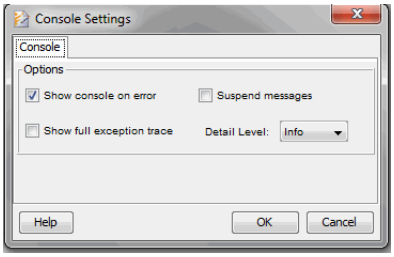
ST.2 Using the Console

If an error occurs in Style Studio, the **Console** panel may open to show relevant exceptions or log messages.



To open the **Console** panel at any time, select ‘Console’ from the ‘Window’ menu. To clear the contents of the **Console** panel, right-click anywhere in the panel itself, and select ‘Clear’.

To set Console properties, select ‘Console Settings’ from the ‘Window’ menu. This opens the ‘Console Settings’ dialog box.



The ‘Console Settings’ dialog box provides the following options:

- ‘Show console on error’ opens the **Console** panel when an error occurs during report preview, query preview, etc.
- ‘Show full exception trace’ generates an execution trace that may be useful in identifying the source of the error.
- ‘Suspend messages’ prevents subsequent log messages from being displayed in the **Console** panel.
- ‘Detail Level’ specifies the level of detail supplied in the log messages. There are five available detail levels, described below:

DETAIL LEVEL	DESCRIPTION
Finest	Very fine-grained performance information.
Fine	Debugging messages.
Info	Non-critical information messages.
Warning	Non-fatal errors and critical messages.
Severe	Fatal errors.

It is recommended that you set the detail level to ‘Warning’ or higher.

ST.3 Deploying an Asset

```
Deployment from Style Studio should generally be restricted to
development servers. Deployment to a production server should be
performed by an administrator. See Incrementally Deploying Assets
for details.
```

You can easily deploy a repository asset (data source, data model, Worksheet, query, table style, etc.) to a repository other than the working repository; that is, to a *remote* repository. The deployment feature allows you to transfer an asset together with all additional assets on which the target asset depends.

Before you deploy an asset to a remote repository, verify that the remote repository is configured under the **Repository** tab of the ‘Style Studio Configuration’ dialog box. If the repository is of type ‘Database’ or

'Server', verify that the indicated database or server is ready to accept connections. See *Configuring a Repository* for more information.

ST.3.1 Deploying an Asset to a Remote Repository

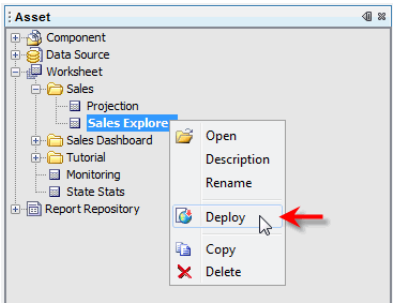
To save an asset to the *working* repository, simply press the 'Save' button in the toolbar.

A *remote repository* is any repository other than the current *working repository*. To deploy an asset to a remote repository, follow the steps below:

1. Open the asset in Style Studio, and press the 'Deploy' button in the Style Studio toolbar. This opens the 'Deploy' dialog box.

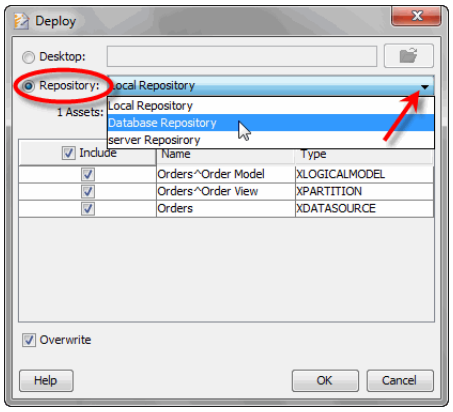
   

   Alternatively, expand the Style Studio **Asset** panel, and right-click the asset that you wish to deploy. Select 'Deploy' from the context menu to open the 'Deploy' dialog box.



See *Configuring a Repository* to configure alternate repositories.

2. In the 'Deploy' dialog box, select the 'Repository' option, and choose the desired remote repository from the menu. (Note that the current working repository is not listed in the menu.)



3. Under the 'Include' column of the table, select the assets that you wish to deploy together with the target asset.

```
Note: Overwriting repository files may interfere with other
dashboards.
```

4. Select ‘Overwrite’ to specify that these deployed assets should replace assets of the same name in the deployment repository. Deselect the ‘Overwrite’ option if you do not wish to modify existing assets.

5. Press ‘OK’ to begin the deployment. Depending on the type of the remote repository (server, database, or local), the deployment may take a few moments.

Assets that you deploy to a remote repository can be managed by an administrator using Enterprise Manager.

ST.3.2 Deploying an Asset to a JAR File

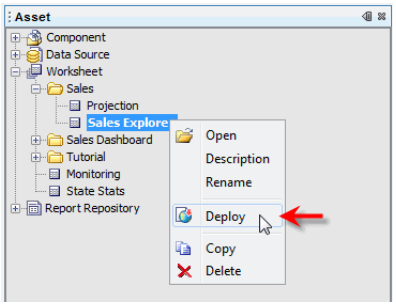
You can deploy an asset to a JAR file on the local file system. This allows you to easily transfer the bundled asset to a different environment at a later time.

To deploy an asset to a JAR file, follow the steps below:

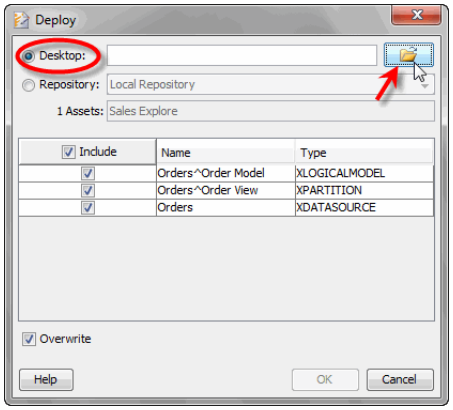
1. Open the asset in Style Studio, and press the ‘Deploy’ button in the Style Studio toolbar. This opens the ‘Deploy’ dialog box.

   Alternatively, expand the Style Studio **Asset** panel, and right-click the asset that you wish to deploy. Select ‘Deploy’ from the context menu to open the ‘Deploy’ dialog box.



2. In the ‘Deploy’ dialog box, select the ‘Desktop’ option, and press the ‘Browse’ button.



This opens the ‘Save’ dialog box.

3. In the ‘Save’ dialog box, specify the location and the ‘File name’ for the exported JAR file, and press ‘OK’.

   The table lists all of the assets that are required by the selected asset. You can choose which of these assets you wish to deploy together with the selected asset.

4. Under the ‘Include’ column of the table, select the assets that you wish to deploy together with the selected asset.

```
Note: Overwriting repository files may interfere with other
dashboards.
```

5. Select ‘Overwrite’ to specify that these deployed assets should replace assets of the same name in the deployment repository. Deselect the ‘Overwrite’ option if you do not wish existing assets to be modified.

6. Press ‘OK’.

This deploys the asset together with any selected required assets, to a JAR file in the location that you specified. You can import the JAR file into an alternate repository by using the ‘Import’ feature of Style Studio or Enterprise Manager in the target repository. See *Importing Assets from a JAR File* and *Incrementally Deploying Assets* for more information.

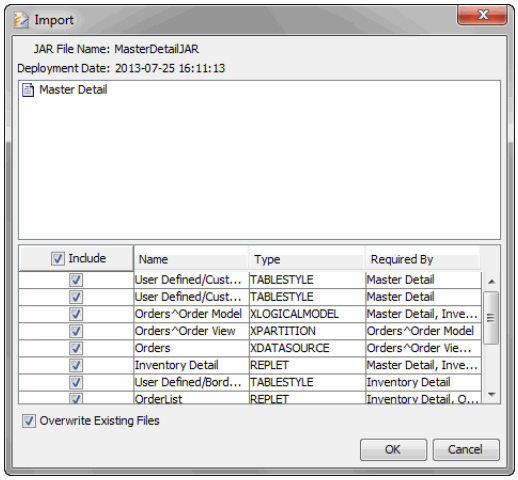
ST.3.3 Importing Assets from a JAR File

You can import an asset from a JAR file that was generated using the Style Studio ‘Deploy’ feature or the Enterprise Manager ‘Export’ feature. See *Deploying an Asset to a JAR File* and *Incrementally Deploying Assets* for more information on creating a JAR file package.

To import assets from a JAR file, follow the steps below:

1. Select ‘Import’ in the Style studio ‘File’ menu. This opens the ‘Open’ dialog box.

2. In the ‘Open’ dialog box, select the JAR file whose contents you wish to import, and press ‘Open’. This opens the ‘Import’ dialog box.



   The items listed in the top panel will all be imported. The items listed in the bottom table are required by items in the top panel, and will be imported by default. However, if one or more of these items already exist in the current repository, you can choose not to import those items.

3. Select the ‘Include’ box next to assets in the table that you wish to import.

`Note: Overwriting repository files may interfere with other dashboards.`

4. Select ‘Overwrite’ to specify that the selected assets should replace assets of the same name in the working repository. Deselect the ‘Overwrite’ option if you do not wish to modify existing assets.

5. Press ‘OK’.

This imports the selected assets into the working repository.

APPENDIX BP: Development Best Practices

This appendix provides a discussion of best practices to use when developing queries, data models, Data Worksheet, reports, and dashboards.

BP.1 Development Workflow

Different applications may require different assets and different approaches, but the most common workflow for creating assets is given below:

1. **Configure Style Studio**



Before you begin creating new assets, etc., make sure that Style Studio is correctly configured. In particular, make sure that the *repository* location is set correctly and that your database driver files are on the classpath. See *Configuring Style Studio* for more information. If you plan to develop assets as a team, see also *Developing as a Team*.

2. **Define datasources**



A *datasource* is a connection to your data, which might be stored in a relational database such as Oracle or SQLServer, in a text or Excel file, or in some other format. By defining a datasource, you provide your reports and dashboards with a *live* connection to your data. This means that when your data is updated, reports and dashboards automatically reflect the updated data. To learn how to create a datasource, see *Connecting to Your Data*.

3. **Create queries, data models, and Data Worksheets**



After you have defined a datasource to connect to your data, you need a mechanism to retrieve the desired data and supply this data to reports and dashboards. You can do this by creating queries, by creating data models, or by creating Data Worksheets. See *Query vs. Model vs. Worksheet* for an explanation of the benefits of each approach.

4. **Create reports and dashboards**



After you have created queries, data models, and/or Data Worksheets, proceed to create your final products: Reports and dashboards. See *Report vs. Dashboard* for an explanation of the benefits of each type of asset.

5. **Deploy reports and dashboards to production server**



After you have created reports and dashboards, deploy them to your production server. See *Deploying an Asset* for more information.

See Also

Using Style Studio, for information how to use Style Studio.

BP.2 Query vs. Model vs. Worksheet

There are three principal ways to retrieve data from a datasource:

- **Query**: A query is a specific request for data from the datasource (i.e., a request for specific columns and rows), typically written in the SQL language. Style Studio provides a wizard to make query creation easy. See *Creating a Query with the Wizard* or *Creating a Query from SQL* for information on how to create a query.
- **Data Model**: A data model provides an all-encompassing representation of your data, a view into your database suitable for business users. Because a data model is not a *specific* request for data, but rather a model of your database schema, it is much more flexible than a query. See *Creating a Data Model* for an example.
- **Data Worksheet**: A Data Worksheet has many roles, but is most often used to graphically build complex queries and mash-up data from different data sources. See *Transforming Your Data* for more information.

The table below highlights the advantages and disadvantages of these different approaches.

FEATURE	QUERY	DATA MODEL	DATA WORKSHEET
Mash up data from different data sources	No	No	Yes
Quick to create	Yes	No	Yes
Create complex queries	Partially (via SQL string)	Partially (via SQL string)	Yes
Facilitate *self-service* (see below)	No	Yes	Partially
Accept SQL string	Yes	Yes	No
Import data from non-datasource	No	No	Yes
Available for non-relational databases	Yes	No	Yes

Self-service refers to the ability of an end-user to make their own data selections. Queries and Data Worksheets are limited in this regard, because they provide access only to the fields that they were originally designed to provide (although, of course, a user can choose among those fields). Because a data model provides a broader view into the entire database schema, it gives the user much more flexibility in selecting data and greatly enhances self-service.

BP.3 Report vs. Dashboard

The end result of your development process is a generally a *report* or a *dashboard*. It is important to understand the similarities and differences between these two methods of data presentation. Both reports and dashboards present data, but they provide a very different user experience.

The most profound difference between reports and dashboards is the level of interactivity:

- To create a highly interactive environment, use a dashboard. See *Creating a Dashboard* for more information.
- To present static data in a multi-page format, use a report. See *Creating a Report* for more information.

The table below compares some other features of reports and dashboards.

REPORT	DASHBOARD
Supports multiple pages.	Supports only a single screen, but allows tabbed interfaces.
Supports multiple data sources in a single report.	Allows only one data source per dashboard, but dashboards with different data sources can be nested.
Static, but can be made more interactive through parameterization.	Dynamic and interactive.
Does not support materialized views.	Supports materialized views.
Provides full desktop-based design tools and limited web-based design tools (Ad Hoc editing).	Provides full web-based design tools.
Supports reusability via beans, meta-templates, subreports, and parameter sheets.	Supports reusability via nesting.
Can be viewed under the Portal **Report** tab.	Can be viewed under the Portal **Report** tab and **Dashboard** tab.
Can be exported to a wide variety of formats.	Can be exported to Excel, PDF, or PowerPoint.

BP.4 Developing as a Team

It is common for multiple developers to work on the same set of projects, and Style Intelligence supports such team development in a variety of ways.

To set permissions for dashboards, Worksheets, and folders, see *Repository Permissions* and *Component/Object Permissions* in *Advanced Server Administration*.

Dashboards and Data Worksheets can be created and edited entirely through the web-browser. Therefore, to work as a team on dashboards and Worksheets, developers should simply log into the same server and edit the dashboards and Worksheets using Visual Composer. See *Creating a Dashboard* and *Transforming Your Data* for more information.

Reports, data sources, queries, data models, and reusable assets such as table styles and parameter sheets are created on the desktop using Style Studio. A developer who works on these assets must therefore have an installation of Style Studio. The easiest way to develop these assets as a team is to utilize a *shared repository*. The repository is the location from which Style Studio reads and writes these assets, and by sharing the same repository developers can access and edit the same assets.

There are three different types of repositories, *local*, *database*, and *server*, and any of these can be shared for team development. See *Configuring a Repository* and *Sharing a Repository* for information on how to set up and share a repository. To deploy assets from one repository to another (e.g., from a development repository to a production server repository), see *Deploying an Asset*.

End of Document

Please visit www.inetsoft.com for technical support, software updates, and new releases.

Style Report™, Style Scope™, and Style Intelligence™ are owned and licensed exclusively by InetSoft Technology Corporation, ©1996-2014 InetSoft Technology Corp. All rights reserved. All other products, names, or brands are the trademarks of their respective holders.